视觉算法工程师成长指导手册

出品单位：有三 AI

作者：言有三 汤兴旺 臧小满

起草时间：2018 年 12 月～2019 年 12 月

导读

本手册以深度学习视觉算法工程师为例，为大家建议一条完整的学习路线，完成从外行到内行的整个学习流程。借鉴广泛采用的评级机制，分为 4 个大境界，即白身，初识，不惑，有识。

每一个境界都由浅入深提供10多篇文章对核心知识点进行梳理，并对技术发展的最新水平进行简单介绍和展望。

1. 白身境界

所谓白身境界，就是基本上什么都不会，还没有进入角色。在这个境界需要修行的内容包括：

(1) 熟练掌握 Linux 及其环境下的各类工具的使用

(2) 熟练掌握 Python 及机器学习相关库的使用

(3) 掌握 C++等高性能语言的基本使用

(4) 知道如何获取和整理，理解数据

(5) 掌握相关的数学基础

(6) 了解计算机视觉的各大研究方向

(7) 了解计算机视觉的各大应用场景

(8) 了解行业的优秀研究人员，知道如何获取最新的资讯，能够熟练阅读简单的技术资料

2. 初识境界

所谓初识，就是对相关技术有基本了解，掌握了基本的使用方法。在这个阶段，需要修行以下内容。

(1) 熟练掌握神经网络

(2) 培养良好的数据敏感性，知道如何正确准备和使用数据

(3) 至少熟练掌握一个深度学习框架的使用

(4) 熟悉深度学习模型的基本训练和调参，网络设计

(5) 掌握深度学习各项核心理论技术

(6) 能使用合适的优化准则熟练评估自己的算法

3. 不惑境界

所谓不惑，就是向高手迈进的开始了，在这个境界的重点就是进一步巩固知识，并且开始独立思考。如果说学习是一个从模仿，到追随，到创造的过程，那么到这个阶段，应该跳过了追随，进入了创造的阶段。

如果是在学校读研究生，就要能够发表水平不错的文章，如果是在公司做业务，就要能够提出正确且快速的解决方案，如果是写技术文章，就要能够信手拈来原创写作而不需要参考。

这个阶段需要修行以下内容：

(1) 熟练玩转数据和模型对一个任务的影响

(2) 能够准确的分析出模型的优劣，瓶颈

(3) 对于新的任务能够快速寻找和敲定方案

(4) 拥有各种各样的深刻理解深度学习模型的技能，从可视化到参数分析等等等

(5) 能够优化模型到满足业务的需求，实现工业级落地

(6) 了解行业的最新进展，并在某些领域有自己的独到理解

4. 有识境界

修行到了有识就步入高手境界了。可以大胆地说自己是一个非常合格的深度学习算法工程师甚至是研究员了，在自己研究的领域里处于绝对的行业前沿，对自己暂时不熟悉的领域也能快速地触类旁通。

无论是眼界，学习能力，还是学习态度都是一流水平，时而大智若愚，时而锋芒毕露，当之无愧的大师兄。

以上内容为本公众号号主言有三按照个人职业经验进行评定，非官方标准，不喜勿取。另配套 GitHub 项目 https://github.com/longpeng2008/yousan.ai 供开源框架熟悉，论文导读。

目录

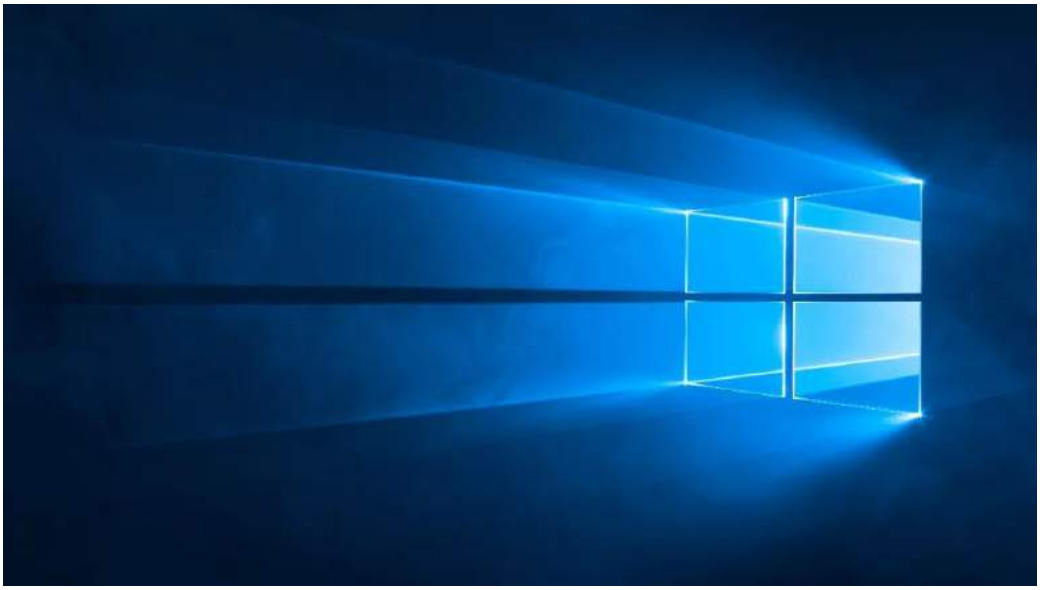
【AI 白身境】深度学习从弃用 windows 开始

本篇是**《AI 白身境》**的第一篇，所谓白身，就是什么都不会，还没有进入角色。给大家准备了 12 期左右的文章来完成这个身份的转变，本篇是第一篇，关于开发环境的选择

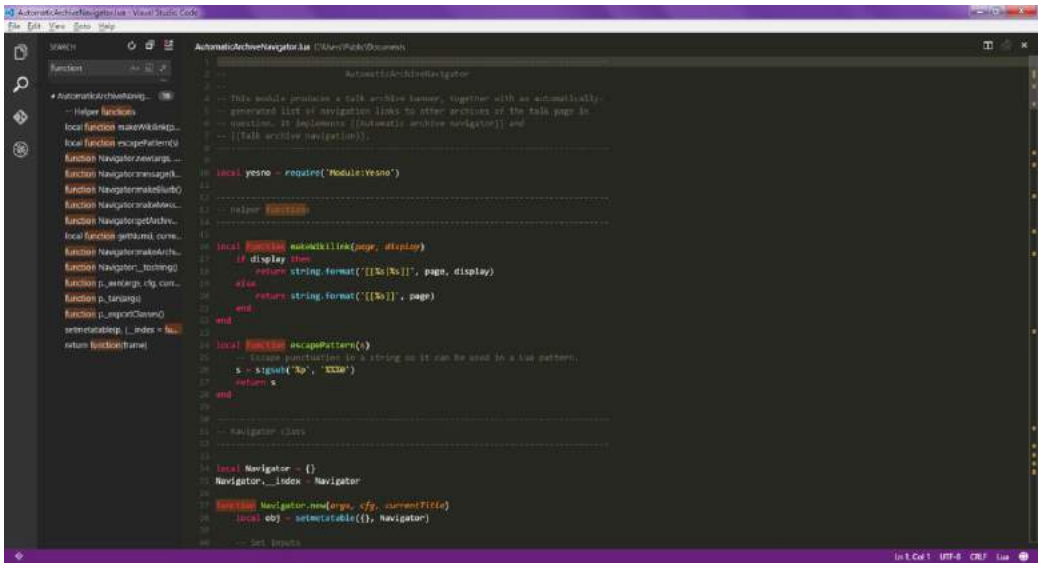
要想正式进入 AI 行业发展，离不开称手的软硬件兵器，其中操作系统就是“软”兵器，本文给大家的建议是**彻底放弃 Windows，只用 Linux 与 Mac**。

作者|言有三

1 windows 写代码不如 linux

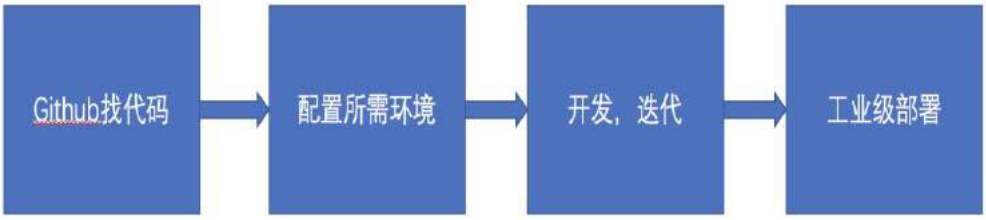


很多人会说，Windows 不适合写代码？各种各样类似于 visual studio 的 IDE 那么牛逼，还不适合写代码？



莫急，且听我说几个理由。

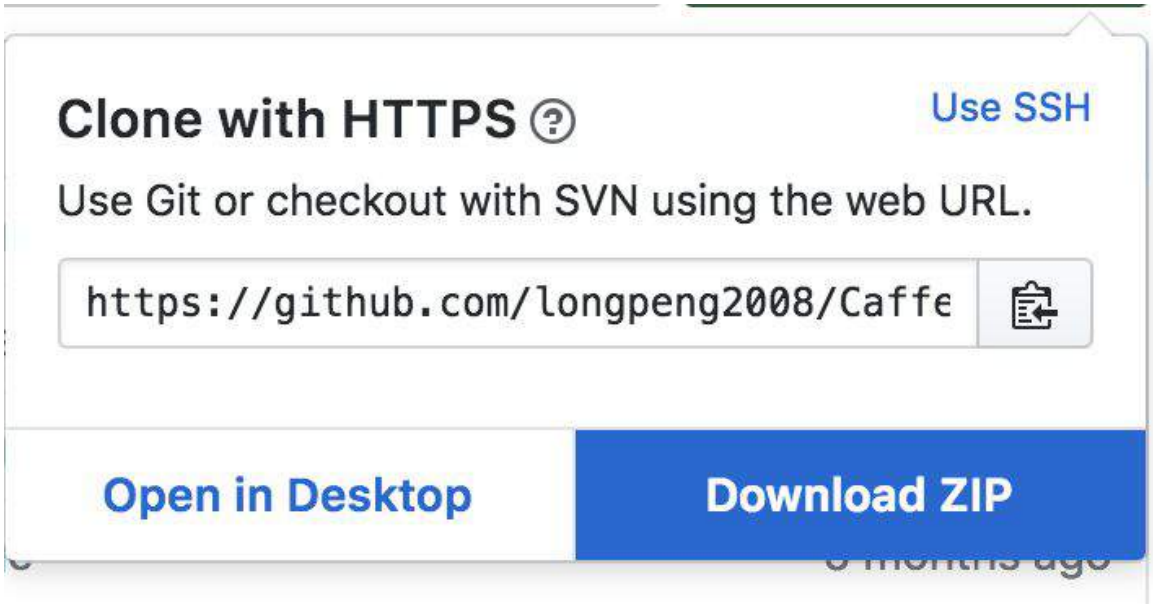
当今**大部分程序员**，开始一个任务时的流程是什么？我想应该是这样的！



我们看看上面这些操作都要干什么。

1.1 github 找代码

最常用的操作包括 git clone，git push，git pull 等，这些在命令行下操作是最简洁优雅的，如果你说每次从 github 上面下载代码采用的是 download 模式，如下：



那么少年，下次我们说 github 的时候好好看看，这样做有多么外行。一句话，windows 不合适。

1.2 配置所需环境

几乎没有一个开源项目是能够下下来直接就能用的，尤其是你的电脑还处于初段水平的时候，**配置环境是新手们的大敌**。

在 **windows 中要装一个新的软件包真的好麻烦**（要自己找软件，看版本，下载安装，配置环境变量），版本控制和更新更麻烦（就是把前面的操作重来一遍嘛），给 python 装各种依赖库好麻烦（想都不敢想），反正就是**很麻烦，巨麻烦，超级麻烦**。而 Linux，通常就是一条命令。

1.3 开发、迭代

这个周期就长了，解决 bug，编译运行等等。看起来，visual studio 之类的 IDE 好像很方便，但是这一切都建立在你还不认识 **VIM** 之类编辑器的强大的前提下。

随随便便说几个功能，比如**列编辑模式，比如复杂的字符替换。**

写起代码来真的是会舒服很多，高效很多。

1.4 模型部署

开发的最后一步就是环境部署，代码经常需要跨平台迁移的，能想象一个依赖于 Windows 下面 IDE 的项目能够毫无隐患，顺利地迁移到嵌入式平台吗？

你很可能不自觉写了一些依赖于 Windows 窗体之类的代码(比如有人喜欢用 C#，MFC)，目录可能也是不上心胡乱配置的，更别说各种日志，到时候就等着重写代码吧。

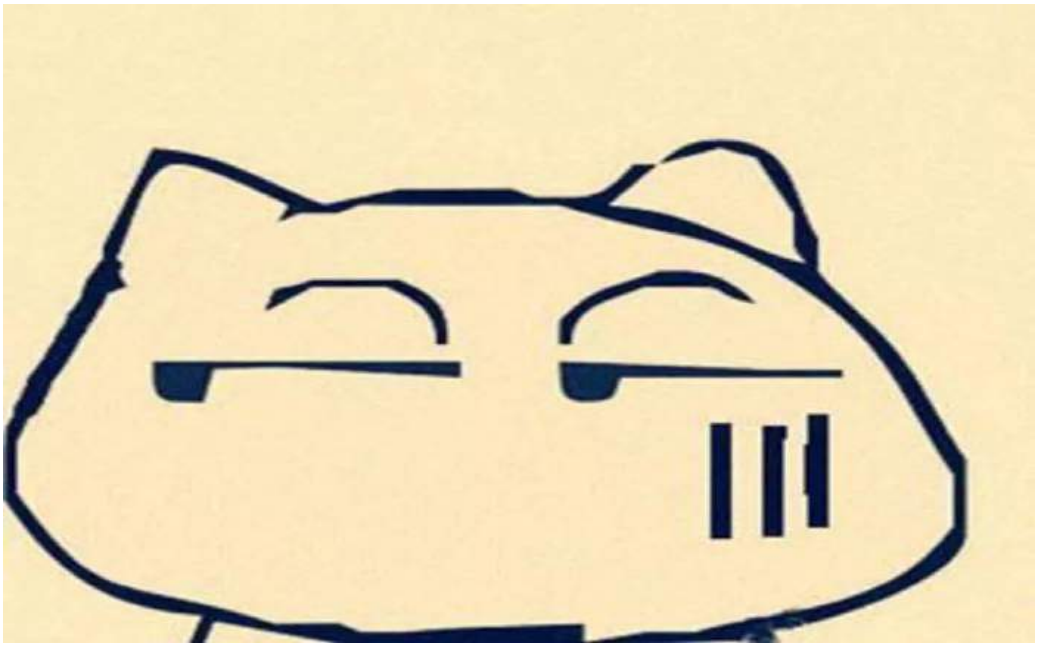
2 windows 没有 Linux 干净

Windows 本就是一个**桌面级的应用系统，不是开发环境**。Windows 是给普罗大众用户用的，不是给程序员用的。是开发好了软件给你用的，而不是开发软件的。

2.1 诱惑太多，没有仪式感

将这个放在第一条貌似有点喧宾夺主，但实际上我觉得是最重要的。

在 Windows 下面搞开发，**写着写着就不知道干什么去了**，反正不写代码也不会死。



在 windows 下面搞开发就没有仪式感好吗！居然还用鼠标，说出去逼格都降低了。

总之，Windows 下开发效率很低，处于鄙视链底层。

2.2 多用户

Linux 实现了用户之间完全的隔离，在同一台机器上，每个人可以有自己独立的目录，如/home/zhangsan，/home/lisi，除非有 root 权限，否则一个用户是看不到别人目录的东西的。

除了公共的软件库和硬件资源，大家在同一台机器上**既可以相互协作，又互不干扰**，这是 Windows 办不到的。

它带来的好处很明显，有几个突出的；

(1) 可以**各自配置独立的环境**，你喜欢 python2，我喜欢 python3，互不侵犯，尊重个性嘛。

这一点非常重要，而一些小团队仍然不重视这个问题。还可以通过配置不同的权限，让小白们权限低一点，老司机们权限高一点，避免出现**小白手贱滥用 apt-get 之类的命令随意更改系统软件库，造成系统崩溃**的情况。

(2) 合理利用资源，比如小实验室买了一块 24G 显存的卡，买不起第 2 块了，总不能放在 Windows 下面分时用吧。

2.3 高效率

Linux 没有复杂的桌面渲染，能更专注地将**服务器的硬件优势表现出来**，有各种各样的命令来进行检测。

对于从事**计算密集型的深度学习算法工程师**来说，GPU 就是命，硬盘都是钱呐。8G 显存，恨不能用到 7.99G。

3 windows 没有 linux 靠谱

这要从两方面来说。

第一是安全，linux 系统是开源系统，人多力量大，bug 往往都被及时发现了。平时很少听到 Linux 中毒的，Windows 在早些年动不动就中毒了。

第二是稳定，Windows 和 Mac，死个机什么的就是毛毛雨，家常便饭习以为常。但是 Linux 突然死机是很罕见很罕见的，我见过最多的就是小白手贱把系统搞死了，类似于 rm -rf /这种。

哪有人这么傻直接运行 rm -rf /，当时是手指在高速运行敲代码，删除其他东西的时候，**不小心带上了**，然后就.....

以上理由，已经足够让你放弃 Windows 了，更多 Linux 的好处，用着用着，就会知道了。

4 总结

长痛不如短痛，如果有做开发者的觉悟了，就尽快换上 Linux 吧。

【AI 白身境】Linux 干活三板斧，shell、vim 和 git

本篇是**《AI 白身境》**的第二篇，所谓白身，就是什么都不会，还没有进入角色。

上一篇给大家介绍了要想真正进入深度学习这个行业，必须要先学会使用 Linux，今天就和大家说说我们应该如何使用 Linux，如何利用 shell，vim 和 git 这三大神器。

作者|汤兴旺 言有三

1 Linux 基础命令与 shell 脚本

通过第一篇的介绍，我们已经知道在 Linux 下面操作会比 windows 下效率高很多，下面和大家讲解一下 Linux 的基础操作，默认大家已经装好了 Linux 系统。

1.1 cd 命令

命令格式：cd〈路径〉

意义：cd 是 change directory 的缩写；cd 命令后面跟一个路径，用于切换当前用户所在的路径，其中路径可以是绝对路径也可以是相对路径。

示例：

cd /system/bin 表示切换到/system/bin 路径下。

cd logs 表示切换到 logs 路径下。

cd / 表示切换到根目录。

cd ../ 表示切换到上一层路径。

1.2 ls 命令

命令格式：ls〈参数〉〈路径〉

意义：ls 是 list 的缩写；ls 命令后面可以跟一个路径或参数，也可以不跟，表示列出路径或当前目录下的所有文件信息。最常用的的参数是“-l”，也就是“ls -l”命令。

示例：

ls / 显示根目录下的所有文件及文件夹。

ls -l /data 显示/data 路径下的所有文件及文件夹的详细信息。

ls -l 显示当前路径下的所有文件及文件夹的详细信息

ls *l wc 显示当前目录下面的文件数量。

1.3 cat 命令

命令格式： cat〈文件〉

意义：cat 是 concatenate 的缩写。表示读取文件内容及拼接文件。

示例：

cat /sys/devices/system/cpu/online 读取 /sys/devices/system/cpu/路径下 online 文件内容。

cat test.txt 读取当前路径下 test.txt 文件内容。

1.4 rm 命令

命令格式： rm〈文件〉或 rm -r〈文件夹〉

意义：rm 是 remove 的缩写。用于删除文件或文件夹，常用参数-r -f，-r 表示删除目录，也可以用于删除文件，-f 表示强制删除，不需要确认。同样的，删除文件前需保证当前用户对当前路径有修改的权限。

示例：

rm -rf path 删除 path。

rm test.txt 删除 test.txt。

1.5 mkdir 命令

命令格式： mkdir 文件夹

意义：mkdir 是 make directory 的缩写。用于创建文件夹。创建文件夹前需保证当前用户对当前路径有修改的权限。

示例：

mkdir /data/path 在/data 路径下创建 path 文件夹。

mkdir -p a/b/c 参数 -p 用于创建多级文件夹，这句命令表示在当前路径下创建文件夹 a， 而 a 文件夹包含子文件夹 b，b 文件夹下又包含子文件夹 c。

1.6 cp 命令

命令格式： cp 〈文件〉<目标文件>或者 cp -r<文件夹><目标文件夹>

意义：cp 是 copy 的缩写。用于复制文件或文件夹。

示例：

cp /data/logs /data/local/tmp/logs 复制/data 路径下的 logs 到 /data/local/tmp 路径下。

cp 1.sh /sdcard/ 复制当前路径下的 1.sh 到/sdcard 下。

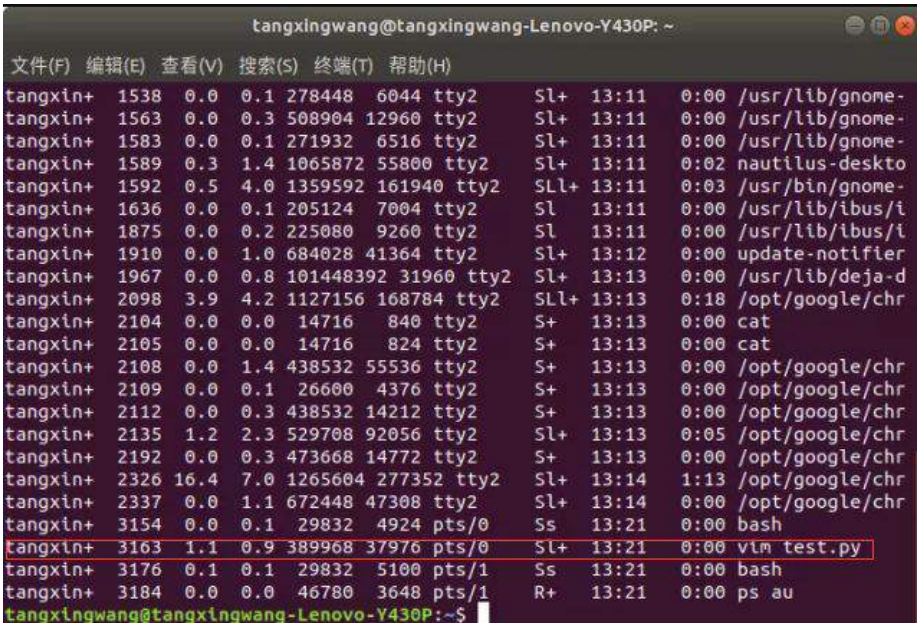
1.7 kill 命令

命令格式：kill PID 码

意义：结束当前进程

示例：

先通过输入命令 ps au 查看进程，找到需要终止进程的 PID 再通过 kill PID 即可，如我这里想要终止的进程是 vim test.py，查到的 PID 是 3163，我们可以输入 kill 3163 结束这个程序，如果结束不了，可以通过 kill -9 PID 码强制结束，即 kii -9 3163



除了这七个命令，还有许多常见的命令，如 pwd 命令，这个可以查看当前路径，这个在移动数据集或者整理文件 list 的时候很有用；tar 命令，这个可以文件压缩；unzip 命令，这个可以用于文件解压，这样的命令其实还有很多，需要我们在使用的过程中不断熟练，需要我们不停的查阅学习。

1.8 shell 脚本文件之"hello world"

有了基本的命令之后，接下来就可以写一些常用的脚本。脚本常用于获取参数，循环遍历。

首先我们看一个“hello world”。

```
#!/bin/sh

a="hello world!"
num=2
echo "a is : $a num is : ${num}nd"
```

运行结果：

```
a is : hello world! num is : 2nd
```

可以看出，用$来获取变量值，通常运行脚本的时候，可以用$1，$2，$3 等获取多个参数。

比如脚本 test.sh

```
x=$1
y=$2
z=$3
echo $1 $2 $3
```

调用的时候就可以：sh test.sh 1 2 3

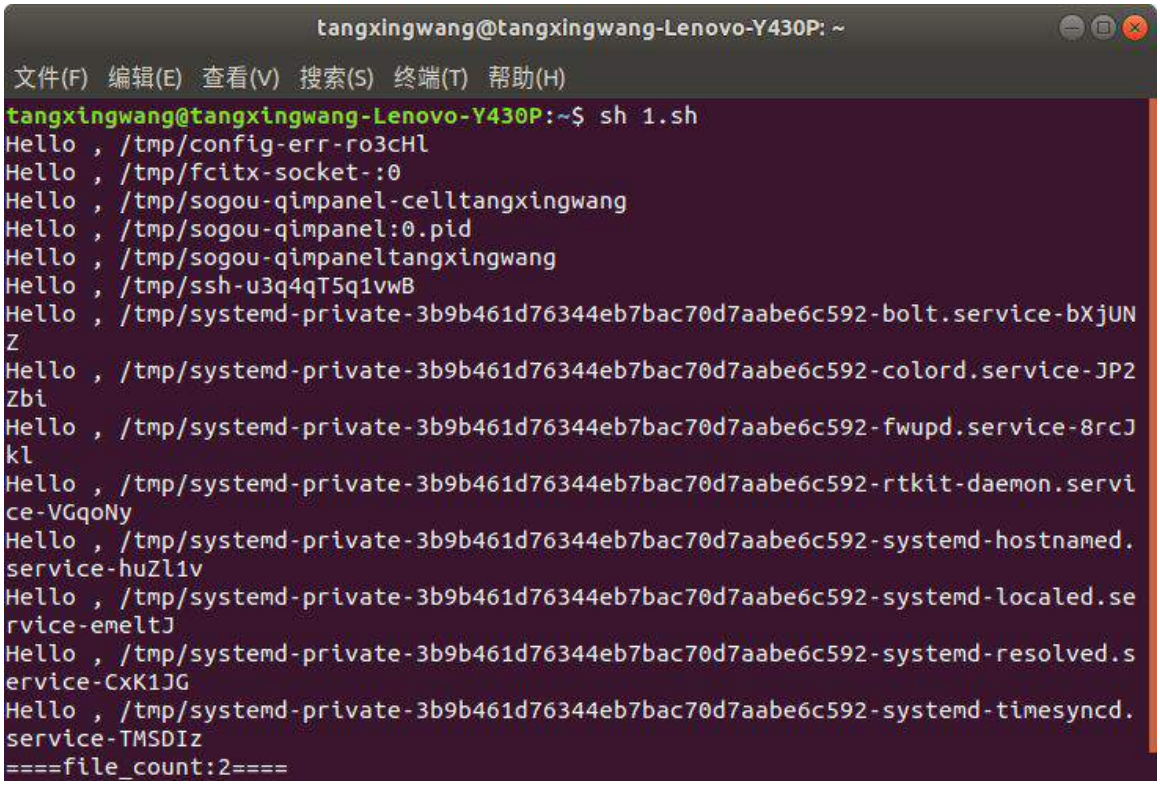
1.9 shell 脚本文件之遍历目录

问题：

1. 切换工作目录至/tmp
2. 依次向/tmp 目录中的每个文件或子目录问好（Hello,log）
3. 统计/tmp 目录下共有多个文件，并显示出来

```
#!/bin/bash
cd /tmp
for  i  in  /tmp/*
do
        echo  "Hello , $i"
done
count=`ls -l|grep  '^-'|wc -l`
echo  "====file_count:$count===="
```

运行结果：



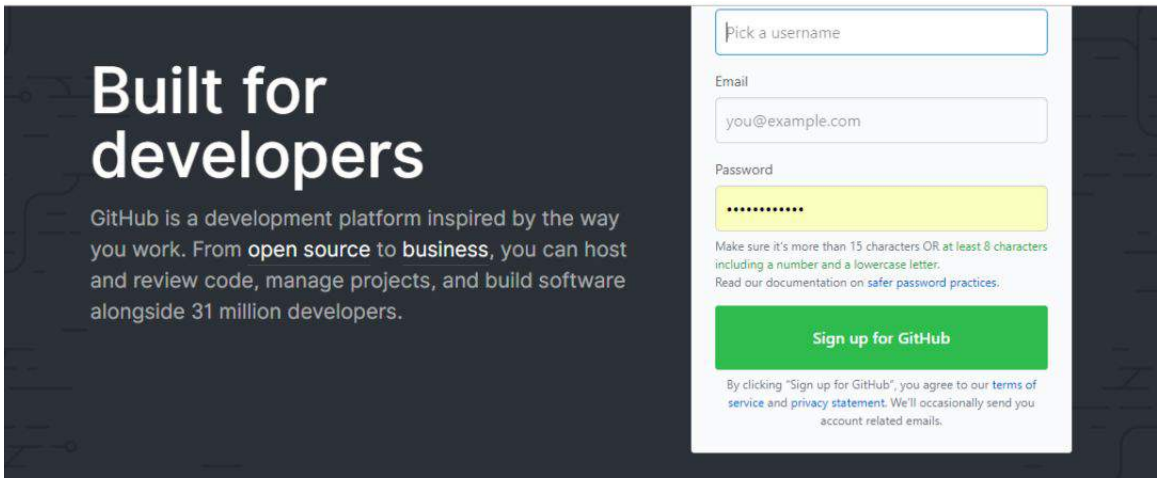
shell 命令还有很多高级用法，大家在实战中进行练习提高吧。

2 github

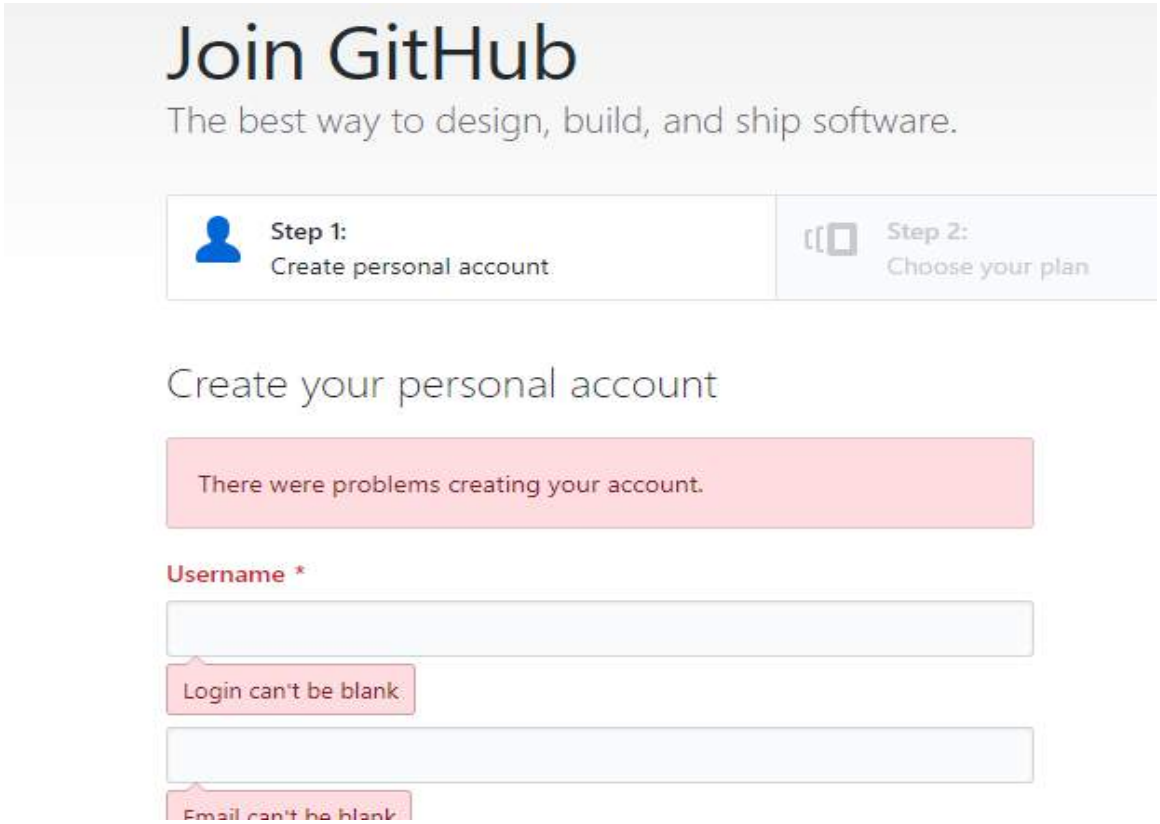
github 是全球最大的程序员交友平台，所以如果你要想从事技术行业，就必须拥有一个账号，跟微信一样离不开你的生活。

2.1 注册 github

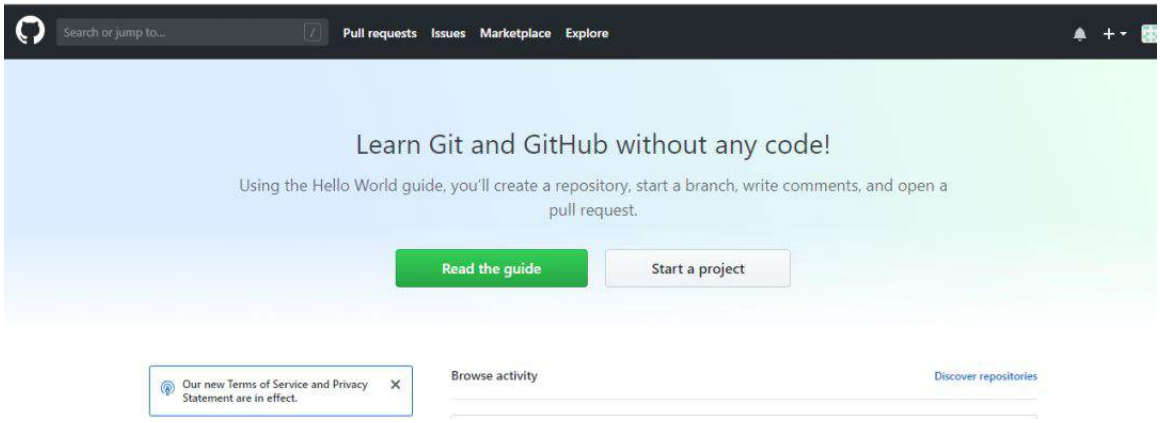
https://github.com/这个是 github 官方网站，我们可以在官网上注册属于自己的 gitHub 账号。点击网址后，界面如下图，由于我们没有 github 账户， 我们需要点击 Sign up for Github 进行简单的注册。



点击 Sign up for Github 后，进入下方这个界面，在 Step1 中填写好个人信息，Step2-3 全部采用默认设置，即可完成 github 注册，记得要去自己的邮箱 verify，不然后面没办法创建仓库。是不是很简单。

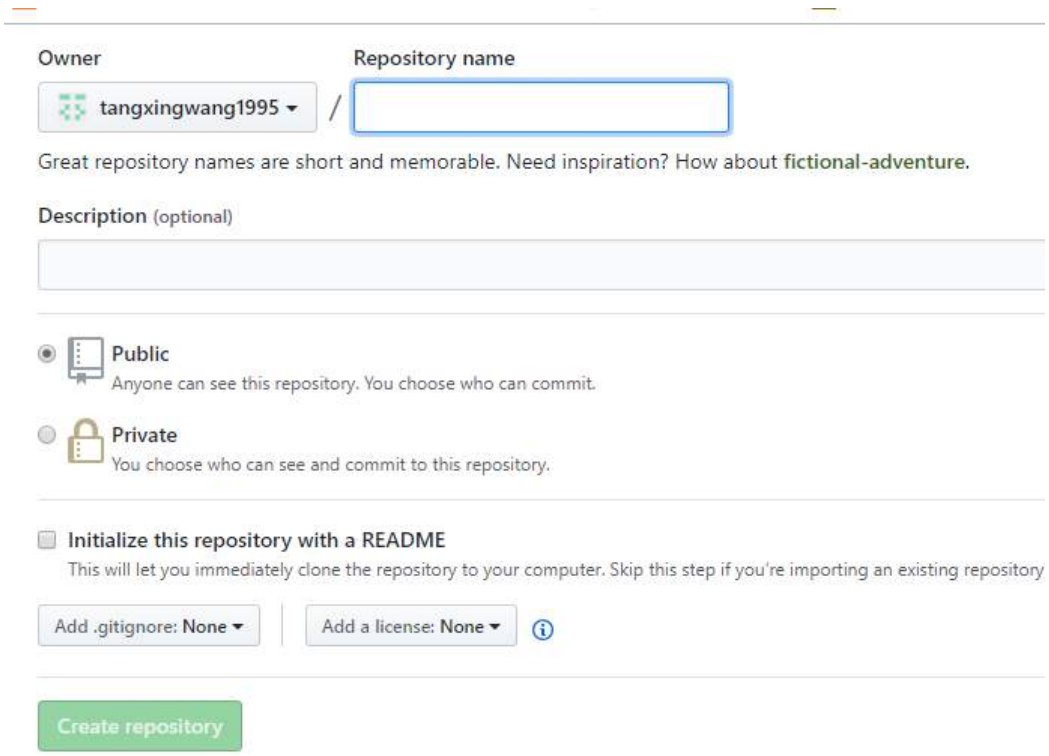


完成上面的步骤后你就拥有了自己的 github 账户，下图就是我的 github 主页。

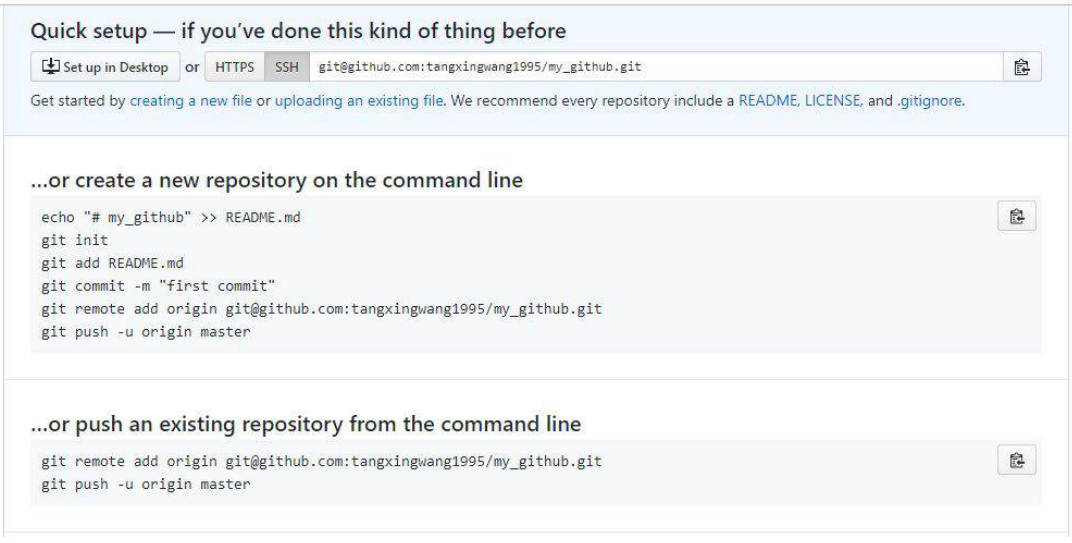


2.2 创建仓库

有了上面的主页后，我们点击 start a project 后就可以创建仓库了，下图就是仓库需要填写一些信息的界面。



仓库名通常就填写我们的项目名，为了说明，这里我填写 my_github，描述可以对自己的仓库进行一个简单的说明，也可以不填。点击“Create repository”按钮，就成功地创建了一个新的 github 仓库，如下图所示：



现在我们就有了自己的 github 和仓库，为了便于管理，我们需要安装一个软件 git。

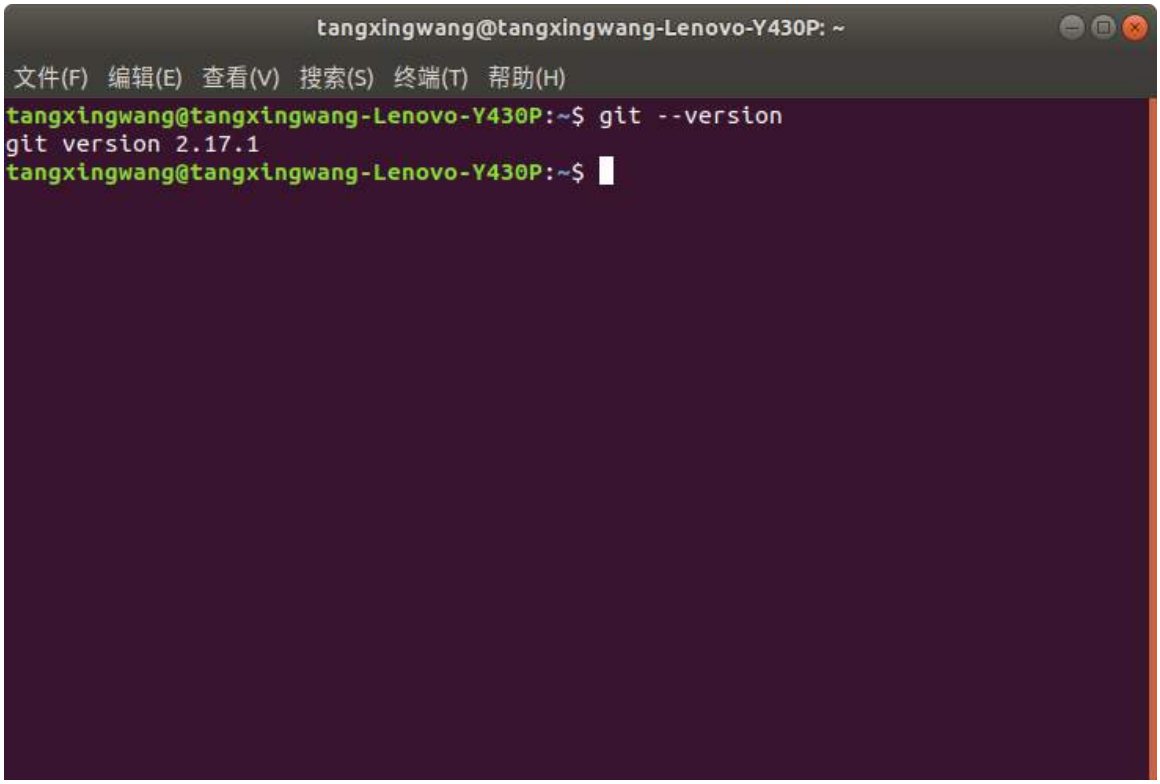
2.3 安装 git

下面我将说一下在 ubuntu18.04 上安装 git，其他的 linux 系统其实也是一样的，安装方法很简单，输入下面命令即可安装。

```
sudo apt install git
```

安装完成后，你可以用下面命令查看 git 版本。

```
git --version
```



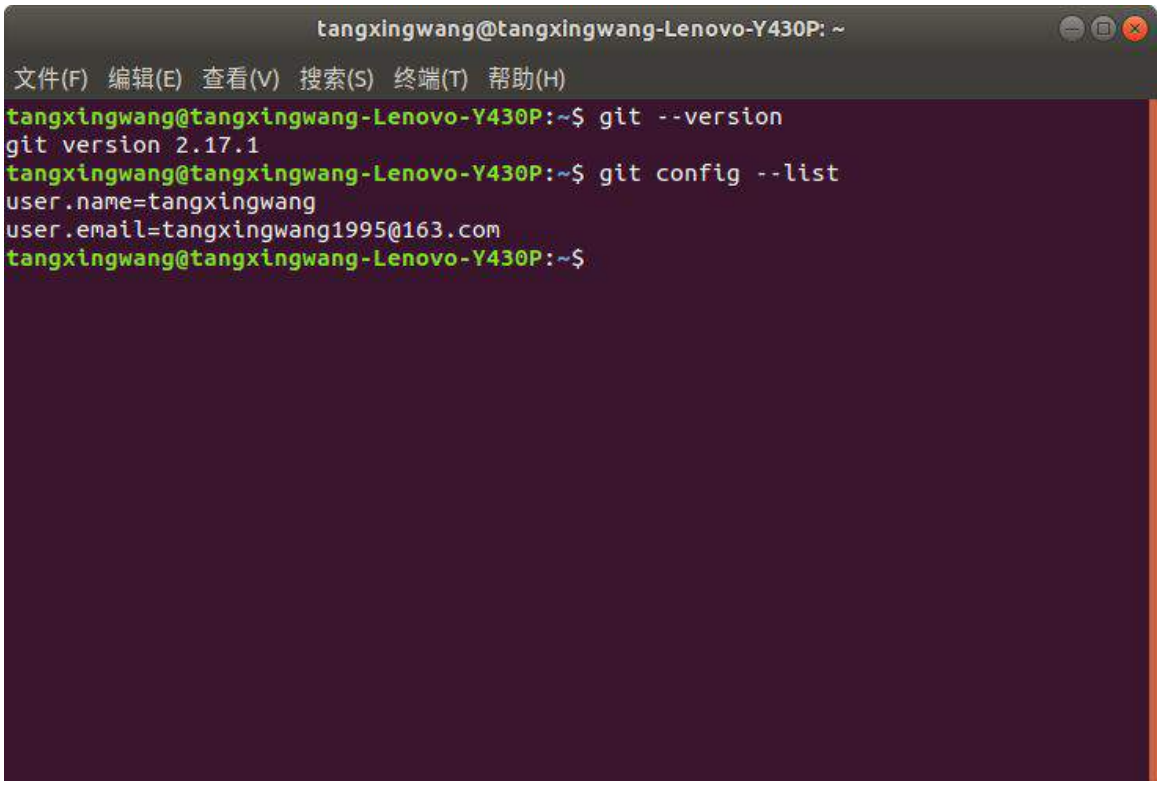
2.4 配置参数

接下来你需要做的就是在 git 中配置自己的名称和电子邮件地址，可以通过使用以下命令来完成此操作：

```
$git config --global user.name "your name"
$git config --global user.email  "your email"
```

我们可以通过下面命令查看是否正确配置。

```
git config --list
```

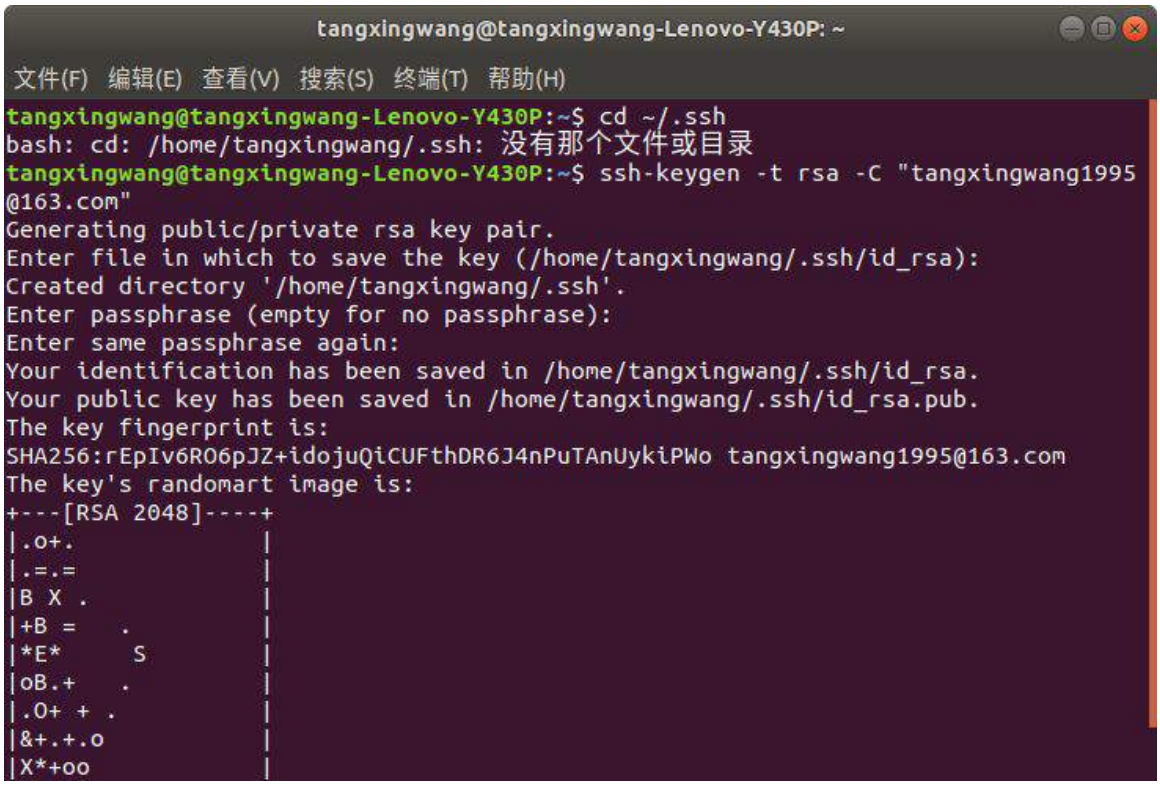


这还没有完，我们还需要创建一个 ssh key，这个实际上就是一个将你的电脑和 github 账号联系在一起的密钥，这样以后就可以十分方便的通过 git 上传你的代码。下面介绍一下如何获得这个密钥，又是如何输入到你的 Github 中。

获取密钥的方法如下：

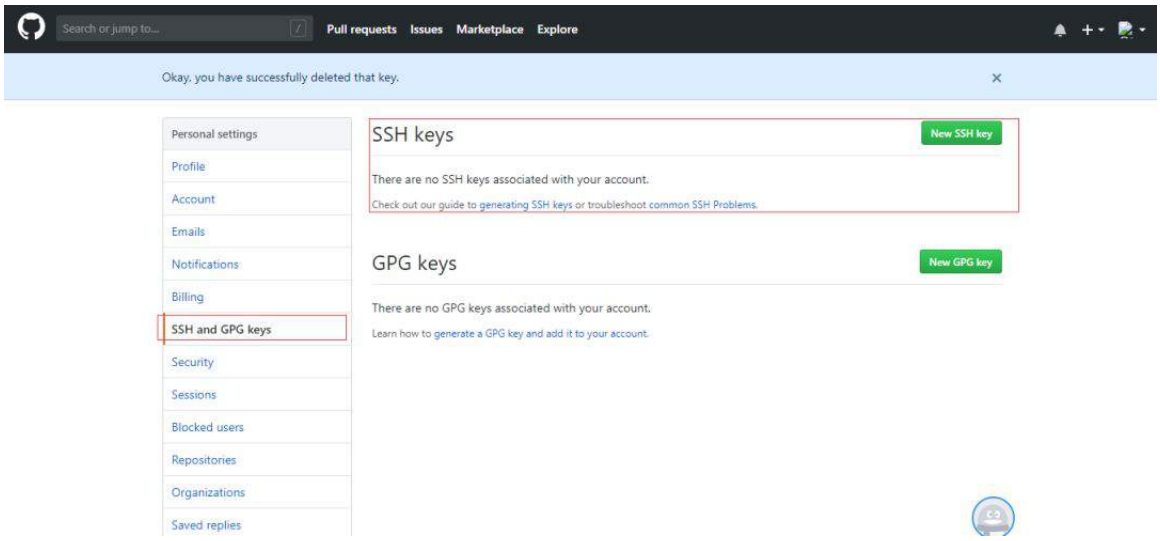
首先在命令行输入 cd ~/.ssh，第一次配置会显示没有那个文件或目录，这是正常现象。然后在命令行输入 ssh-keygen -t rsa -C "邮箱地址"，接下来连按三次回车就可以了。

命令行代码如下：

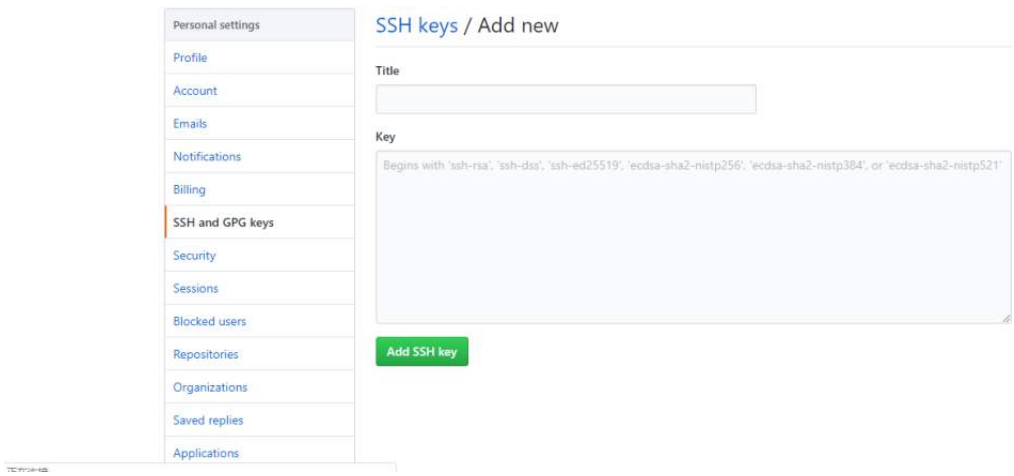


这样我们的密钥就创建成功了。

然后打开/home/tangxingwang/.ssh/id_rsa 文件夹下 id_rsa.pub 文件，复制里面的内容，打开之后不要惊讶，这就是你需要的密钥。你需要登录你的 github 来添加这个密钥，登录 github 后找到 SSH and GPG keys 这个选项（在 setting 里面），然后点击网页右上角的 New SSH keys 进行添加。具体细节如下图:



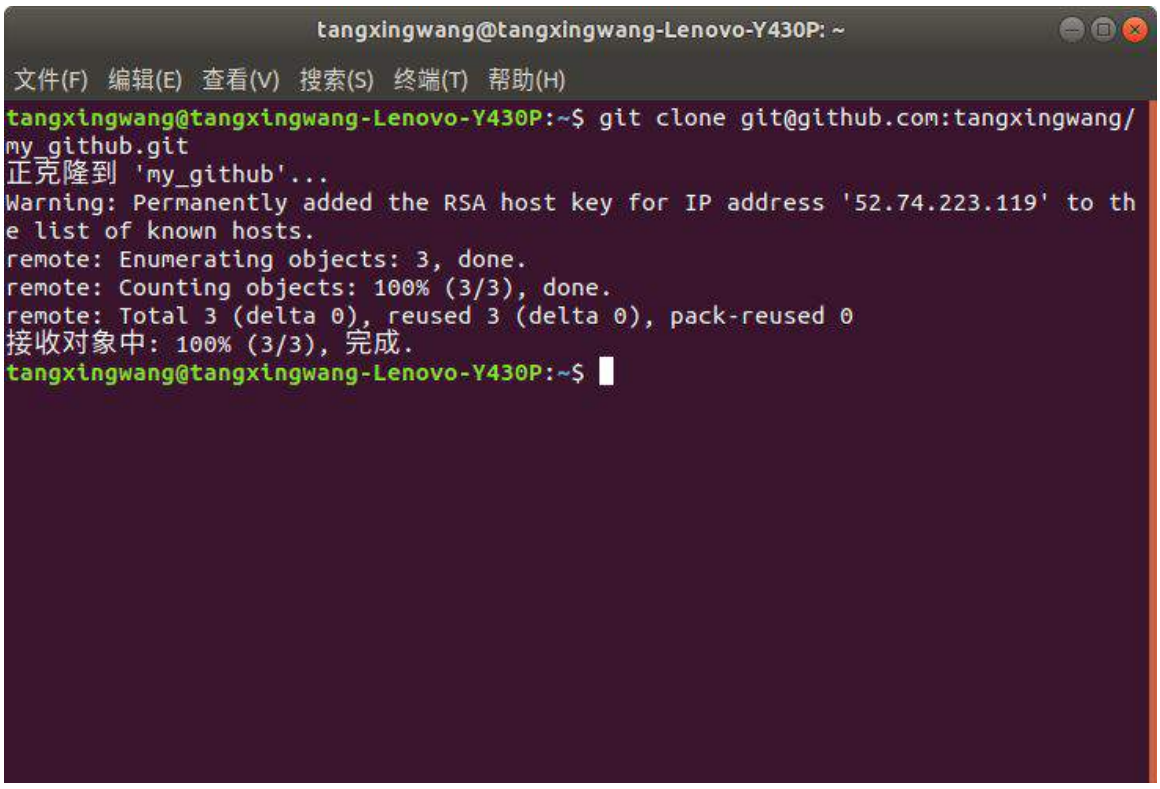
点击 New SSH keys 后界面如下图所示，这里的 Title 是让你给你的密钥起一个名字，随便起一个就行，然后把你刚刚复制的密钥填写在下边的大框里，点击 Add SSH keys 即可。



2.5 clone 操作

当我们想要从 github 上面拉取代码时，就需要使用 clone 操作，现在我们看看怎么进行 clone，其实很简单，只需要输入 git clone<需要 clone 的地址>，示例如下：

```
git clone git@github.com:tangxingwang/my_github.git
```



这样就 clone 成功了，是不是很简单。

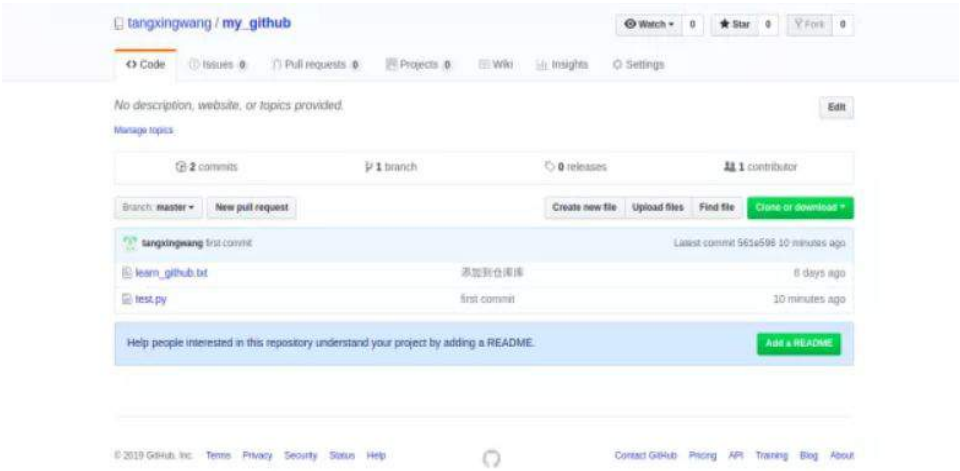
有的时候我们需要拉取依赖库，就需要加上--recursive 选项。

2.6 push 操作

说完 clone，我们再讲讲 push，现在我想在刚刚 clone 下的文件夹 my_github 里面添加一个新的文件 test.py，然后把它 push 到 github 中。命令如下：

```
cd my_github
touch test.py
git add test.py
git status
git commit -m"first commit"
git push origin master
```

这样我们就 push 成功了，我们再看看 github



3 VIM 基本操作

最后我们说说编辑器之神 vim。vim 是从 vi 发展出来的一个文本编辑器，其在代码补全、编译等方便的功能特别丰富，在程序员中被广泛使用。

3.1 基本命令模式

用户刚刚启动 vi/vim，便进入了命令模式。

此状态下敲击键盘动作会被 vim 识别为命令，而非输入字符。比如我们此时按下 i，并不会输入一个字符，i 被当作了一个命令。

以下是常用的几个命令：

- i 切换到输入模式，以输入字符。

- x 删除当前光标所在处的字符。
- : 切换到底线命令模式，以在最底一行输入命令

3.2 输入模式

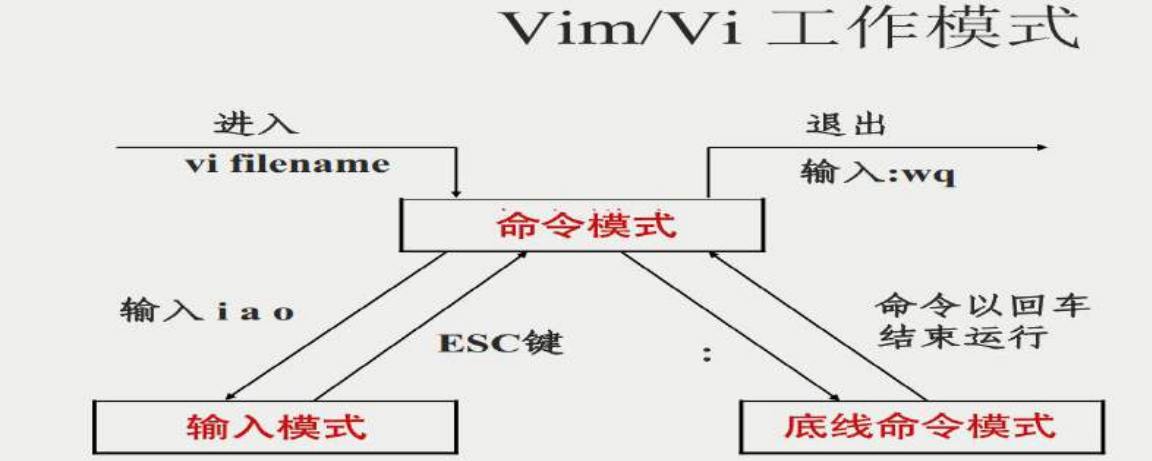
在输入模式下可以对文件执行写操作，类似在 Windows 的文档中输入内容。进入输入模式的方法是输入 i、a、o 等插入命令，编写完成后按 Esc 键即可返回基本命令模式。

3.3 底线命令模式

如果要保存、查找或者替换一些内容等，就需要进入底线命令模式。

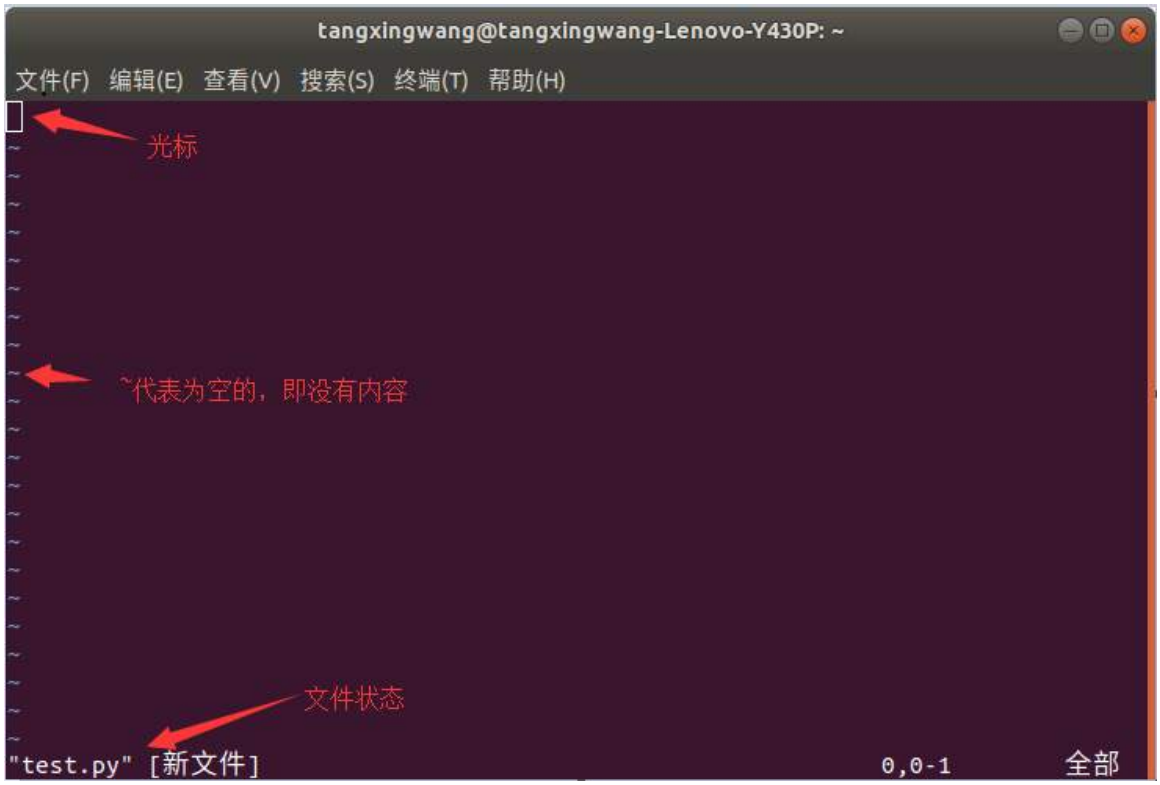
底线命令模式的进入方法为：在基本命令模式下**按":"键**，vim 窗口的左下方会出现一个":"符号这时就可以输入相关的指令进行操作了。

对于新手来说，经常不知道自己处于什么模式，不论是自己忘了，还是不小心切换了模式，都可以按一次 Esc 键返回基本命令模式。如果你多按几次 Esc 键后听到"嘀————"的声音，则代表你已经处于基本命令模式了。



3.4 vim 使用实例

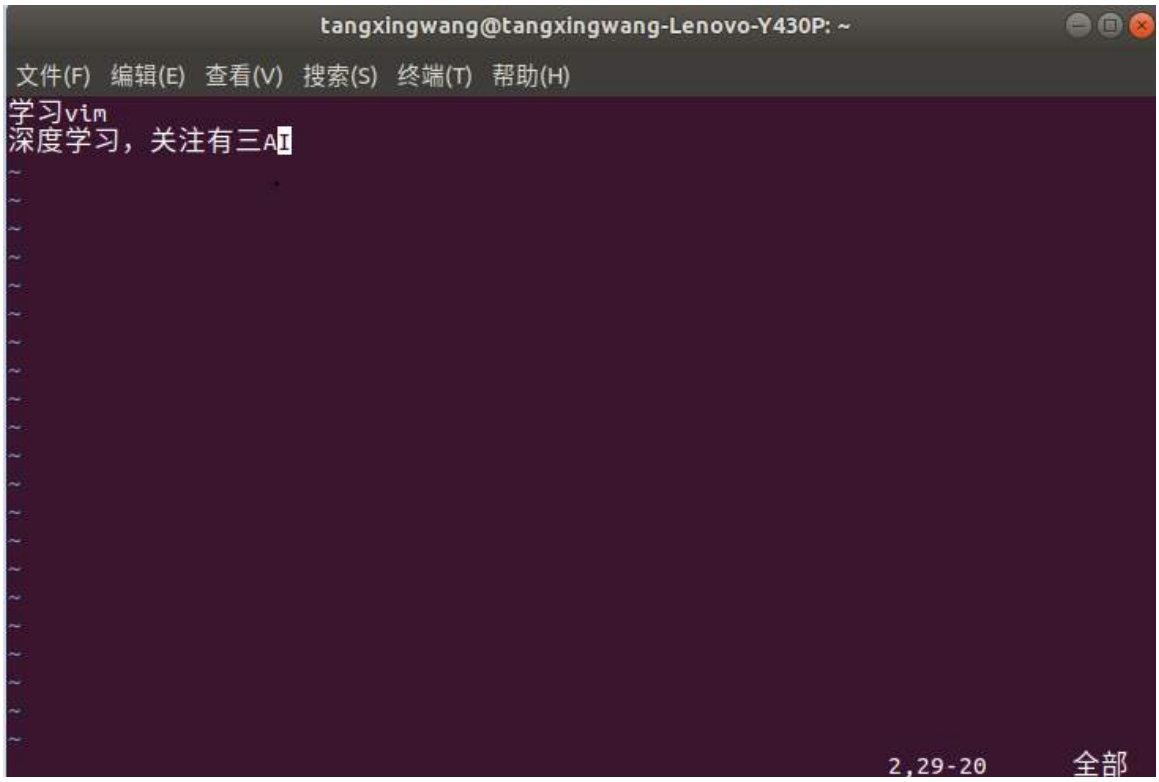
现在我们使用 vim 来建立一个名为 test.py 的文件，你可以这样做：vi test.py，这样就就入了基本命令模式了



按下 i 进入输入模式，开始编辑文字，其实在基本命令模式下，只要按下 i， o， a 等字符就可以进入输入模式了！但各自的功能不同。

其中 i 是光标前插入，a 是光标后插入，o 是换行。另外在输入模式当中，你可以发现在左下角状态栏中会出现 -插入- 的字样，那就是可以输入任意字符的提示。这个时候，键盘上除了 Esc 这个按键之外，其他的按键都可以视作为一般的输入按钮了，所以你可以进行任何的编辑。

那么假设我已经按照下面的样式给它编辑完毕了，应该要如何退出呢？其实很简单，就是给它按下 Esc 这个按钮即可！马上你就会发现画面左下角的 -插入 - 不见了！



对文件编辑完后，我们需要对文件进行保存，其实存盘并离开的指令很简单，在基本命令模式下输入 **:wq** 即可保存离开！

或者按住 shift，连续按两次大写的 ZZ。

OK！这样我们就成功创建了一个 test.py 文件

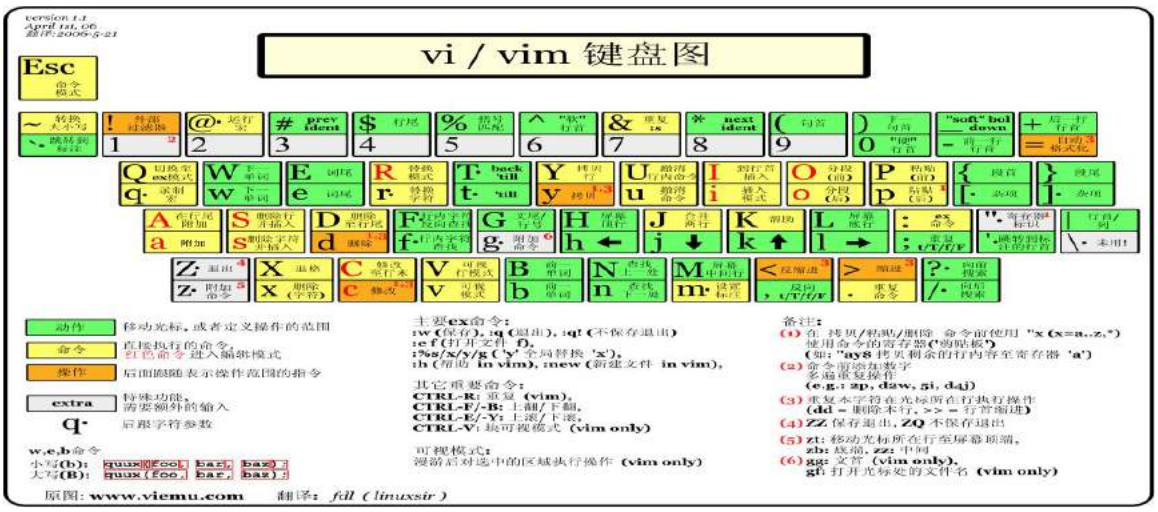
3.5 vim 按键说明

除了上面简易范例的 i， o,a，Esc， :wq 之外，其实 vim 还有非常多的按键可以使用

移动光标的方法	
h 或 向左箭头键(←)	光标向左移动一个字符
j 或 向下箭头键(↓)	光标向下移动一个字符
k 或 向上箭头键(↑)	光标向上移动一个字符
l 或 向右箭头键(→)	光标向右移动一个字符
如果你将右手放在键盘上的话，你会发现 hjkl 是排列在一起的，因此可以使用这四个按钮来移动光标。 如果想要进行多次移动的话，例如向下移动 30 行，可以使用 "30j" 或 "30↓" 的组合按键， 亦即加上想要进行的次数(数字)后，按下动作即可！	

指令行的储存、离开等指令	
:w	将编辑的数据写入硬盘档案中(常用)
:w!	若文件属性为『只读』时，强制写入该档案。不过，到底能不能写入， 还是跟你对该档案的档案权限有关啊！
:q	离开 vi (常用)
:q!	若曾修改过档案，又不想储存，使用 ! 为强制离开不储存档案。
注意一下啊，那个惊叹号 (!) 在 vi 当中，常常具有『强制』的意思～	
:wq	储存后离开，若为 :wq! 则为强制储存后离开 (常用)

这些基本命令需要我们在使用过程中不断的总结，这样才会融会贯通。附上一张 vim 的键盘图，哈哈。



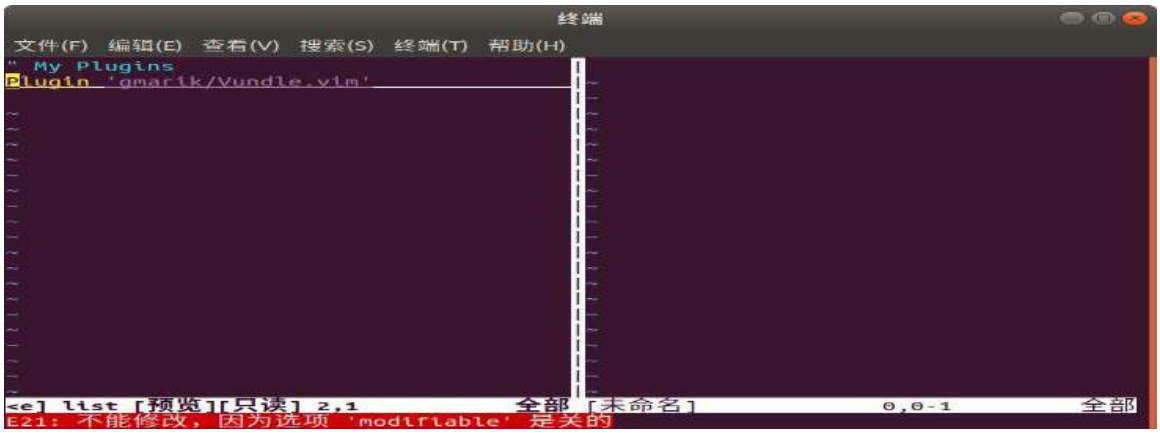
3.6 vim 插件攻略

工欲善其事，必先利其器。一个强大的开发环境可以大大提高工作效率。好吧，我知道这是废话。。。不过，我想一定有很多跟我一样打算进入 Linux 平台开发的新手，一开始都为找不到一个像 Windows 下的 VS 那样可以一键安装并且功能几乎完美无缺的开发工具而郁闷不已，甚至打算收回刚刚迈出的脚步。所幸的是，Vim 有许多强大的插件。

3.6.1 vim 插件之 Vundle

vim 通过插件可以被拓展出不同层次的功能。通常，所有的插件和附属的配置文件都会存放在 ~/.vim 目录中。由于所有的插件文件都被存储在同一个目录下，所以当你安装更多插件时，不同的插件文件之间相互混淆。因而，跟踪和管理它们将是一个恐怖的任务。然而，这正是 Vundle 所能处理的。

Vundle，分别是 vim 和 Bundle 的缩写，它是一款能够管理 vim 插件的非常实用的工具。它为每一个你安装的插件创建一个独立的目录树，并在相应的插件目录中存储附加的配置文件。因此，相互之间没有混淆的文件。简言之，Vundle 允许你安装新的插件、配置已有的插件、更新插件配置、搜索安装的插件和清理不使用的插件。所有的操作都可以在一键交互模式下完成



3.6.2 Vim 插件之 YouCompleteMe

使用 Vim 编写程序少不了使用自动补全插件。这时候当然少不了 YouCompleteMe，它是一个随键而全的、支持模糊搜索的、高速补全的插件。YCM 由 google 公司搜索项目组的软件工程师 Strahinja Val Markovic 所开发，YCM 后端调用 libclang(以获取 AST,当然还有其他语言的语义分析库)、前端由 C++ 开发(以提升补全效 率)、外层由

python 封装,这就是最好用的自动补全插件。vim 插件还有很多，大家可以根据自己的需要进行安装。

当我们真正熟悉使用了上面的 3 类工具之后，就从 Linux 菜鸟开始进步了。

4 总结

你现在对 shell，vim，git 是不是有点感觉了，抓紧学习，也要期待我们下一篇的内容哟。

【AI 白身境】学 AI 必备的 python 基础

本篇是《AI 白身境》的第三篇，所谓白身，就是什么都不会，还没有进入角色。上一篇给大家介绍了如何正确使用 Linux，如何利用 shell，vim，git 这三大神器。相信大家也掌握的差不多了，今天就和大家分享下对于 python，我们应该如何掌握，如何正确把它和深度学习完美的结合起来。

作者|汤兴旺 言有三

1 基础操作

人生苦短，必须学好 python！python 现在火的程度已经不需要我多言了，它为什么为火，我认为有两个原因，第一是人工智能这个大背景，第二是它真的太容易学了，没有任何一门语言比它好上手，接下来我将和大家分享下 python 的基础操作。另外请注意，我的所有操作都是基于 python3！

1.1 python 核心内容之函数

如果你想要学好 python，务必学好 function，不然就相当于没学过 python。

1.1.1 函数定义

在 python 函数定义时有五个要点，分别是 def、函数名、函数体、参数、返回值、以及两个英文版符号：小括号（括号内为参数）和冒号。下面对这 5 点分别解释下：

def：函数关键字。必须要有，系统看到它，就知道下面是个函数了。

函数名：函数的名称。就是给函数起了个名字，当你调用函数时，用函数名就可以直接调用了。

函数体：函数中进行的具体操作。就是你这个函数想要实现的功能。

参数：提供给函数体。

返回值：当函数执行完毕后，可以给调用者返回想要的数据。

下面通过一个具体的实例来说明下：

```
def get_image(picture_path):
        img = cv2.imread("picture_path")
   return   img
```

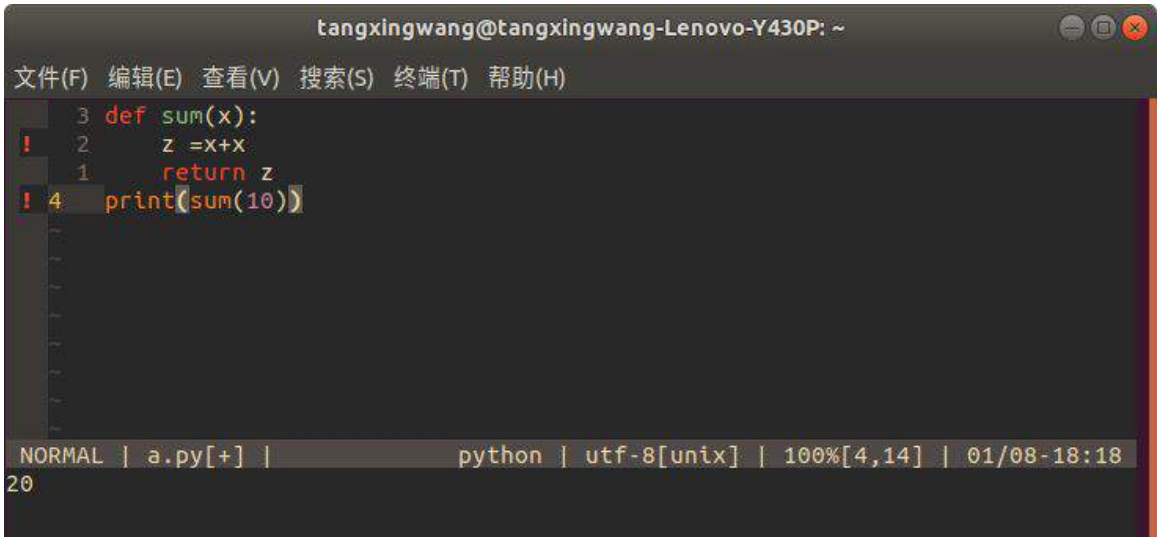
上面实例中，get_image 是这个函数的函数名，这个函数的参数是 picture_path，就是图片的路径，这个参数会传到函数体中。如果你的图片路径是 d://01.jpg，这时候函数体就会变成 img = cv2.imread("d://01.jpg"),最后返回图片。

1.1.2 函数参数

相信你已经知道函数应该如何定义了，接下来再说说函数中最难理解也是最重要的一点，那就是函数参数。

首先我们说说**位置参数**。

```
def sum(x):
    z = x+x
    return z
>>>sum(10)
20
```



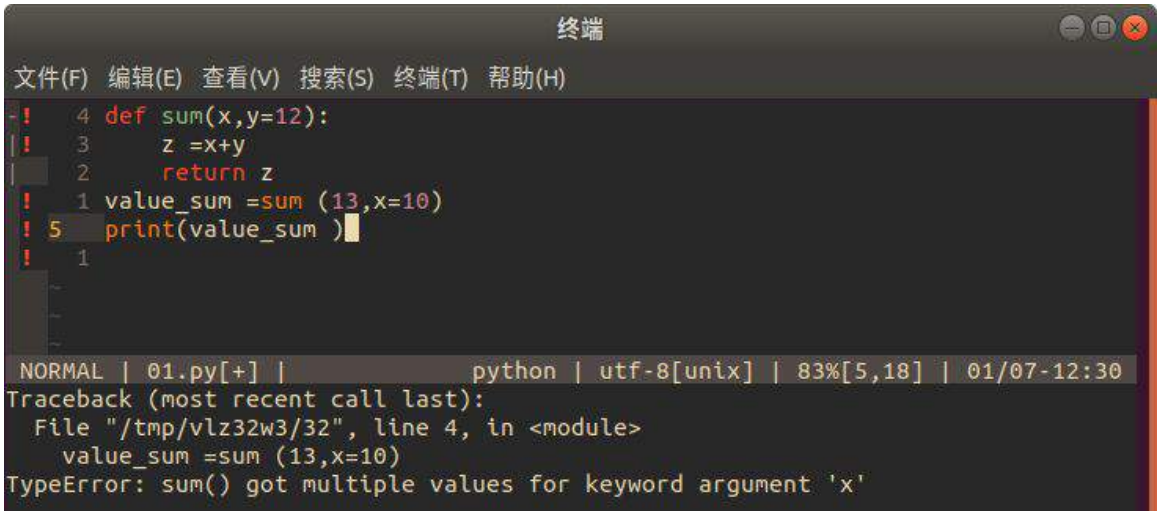
这里的 x 可以认为是一个位置参数，顾名思义，x 先占一个位置，当给予它一个值时，它会传到函数体中，注意像这种位置参数，务必要给予一个值，不然程序会报错。

接下来说说**默认参数**。

```
def sum(x,y=12):
    z = x+y
```

```
   return z
>>>sum(10)
22
>>>sum(10,13)
23
```

这个实例中，y = 12 就是个默认参数，当该参数没有传入相应的值时，该参数就使用默认值。但有点需要注意：默认参数必须在位置参数后面，否则会报错。

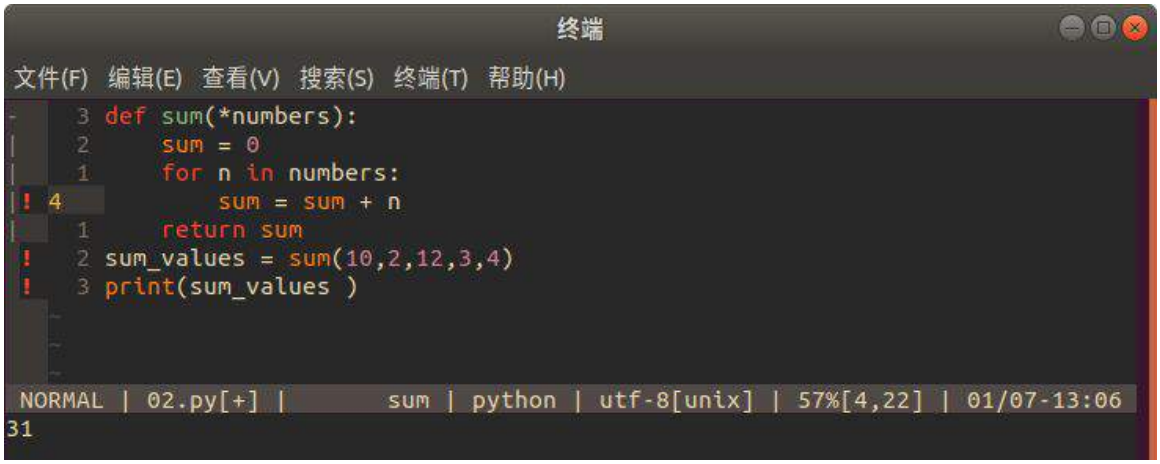


哈哈，报错了，千万不要放这样的错误哟，想避免这样的错误很简单，就从你定义的顺序从前往后写就行。

然后再说说**可变参数**。

在使用 python 函数时，有时候我们不知道我们需要传入多少个参数，于是就有了可变参数的这个概念。为了更好的理解这个概念，先抛出一个问题：计算 a+b+c+d+...的和，因为不知道有几个数字，所以是个可变的问题。那么如何用 python 函数来解决这个问题呢，如下：

```
def sum(*numbers):
    sum = 0
    for n in numbers:
         sum =sum +n*n
 return sum
>>>sum(10,2,12,3,4)
```



我们在参数前面加了一个*号。这样这个参数就变成了可变参数。在调用该函数时，可以传入任意个参数，包括 0 个参数。

最后说一下**关键字参数**。

什么是关键字参数，对于这个概念我们先看下面的代码：

```
def penson(name,age,**kw):
       print('name:',name,'age:',age,'other:',kw)
>>>person('zhang san',24,city='changchun')
name:zhang san age:24 other:{'city':'changchun'}
```

通过上面的例子你应该明白了关键字参数是什么了吧，实际上就是你传入的参数比你之前定义的参数会多，注意位置参数必须要给它传值。

这就是 python 函数的一些基本的方法，更复杂的函数实际上也就是上面的组合而已，只要多加练习，一定能够很好的掌握它。

1.2 python 缩进规则

你可能已经注意到上面我写 python 函数时用到了许多缩进，你可能也会问自己为什么要采用缩进，应该如何缩进这些问题，下面请看我一一道来。

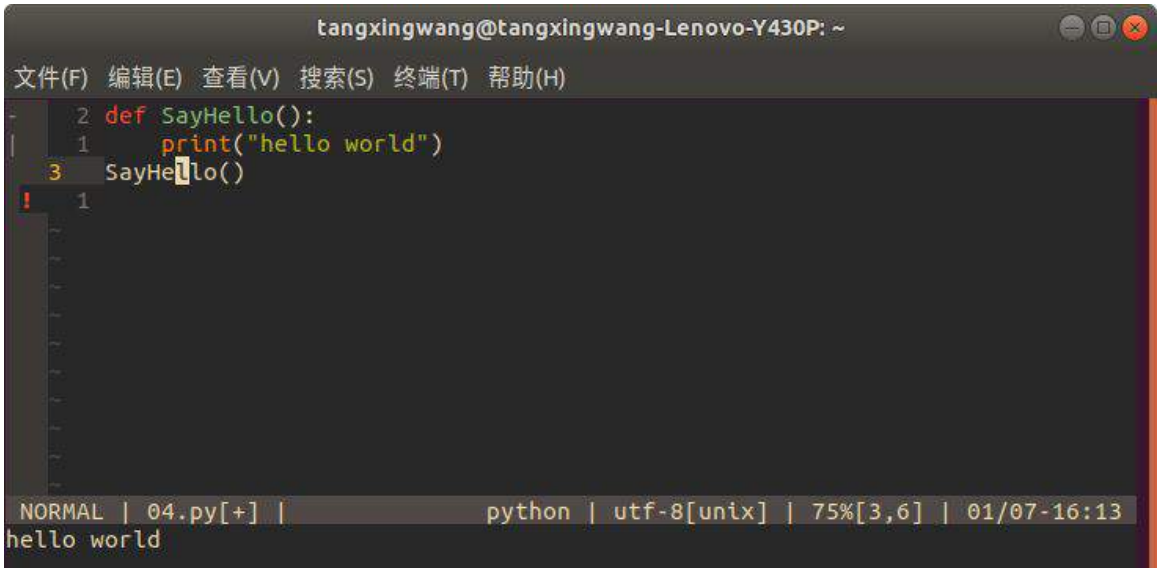
1.2.1 python 缩进的由来

我们在大学时可能都学过 c 语言，在 c 中主要通过{}来区分代码快，但我们初学者往往忘记打{}，而且花括号多了，我们就晕了。而 python 就不会出现这种问题，python 中的缩进可以理解为 c 中的{}。我们来看下下面这个例子：

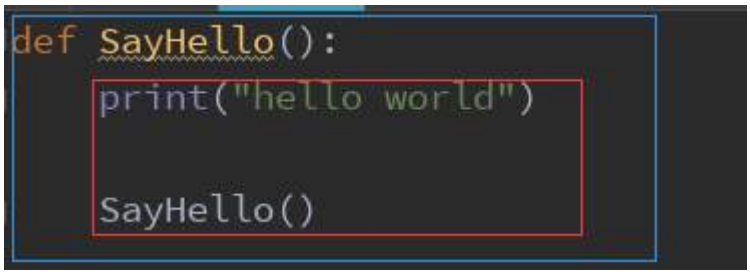
```
def SayHello():
    print("hello world")
      SayHello()
```

你会发现此时不能输出任何结果，我们再看下面一段代码

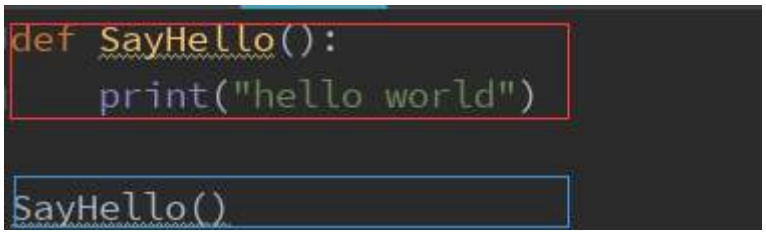
```
def SayHello():
      print("hello world")
SayHello()
```



这样就成功了，为什么会这样呢，下面我介绍一种画框法。如下图所示相同颜色框在一起说明它们是属于同一代码块。



这段代码只是定义了一个函数并未执行它，正确的写法如下：



以后大家可以用这种画框法确定缩进是否正确。

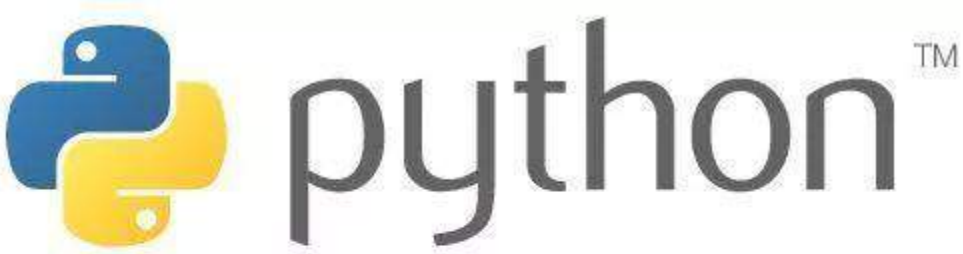
2 矩阵库——NumPy

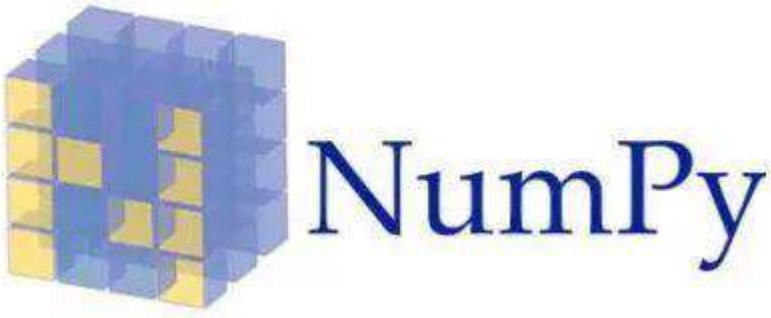
NumPy(Numerical Python) 是 Python 语言的一个扩展程序库，支持高维数组与矩阵运算，提供了大量的数学函数库。

对于深度学习来说，高维数组我们用的很多，因此要想学好深度学习，必须**对 NumPy 了如指掌**。

2.1 ndarray 对象

在 NumPy 中我们用 ndarray 表示数组，可以说它是整个库的核心。下面我们将从以下几个方面来理解 ndarray。

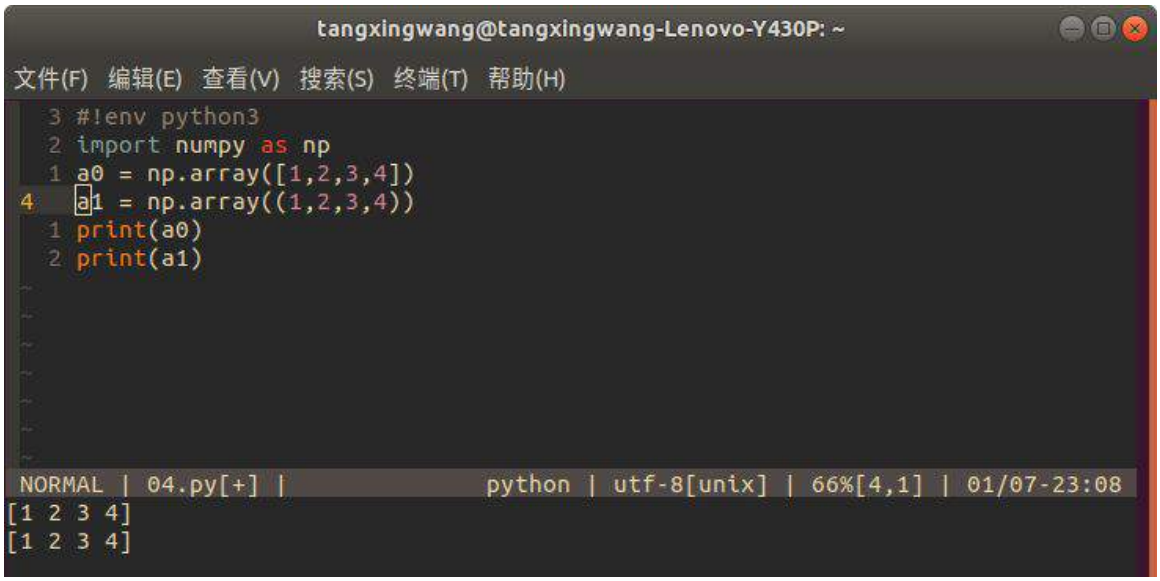




2.2 创建数组

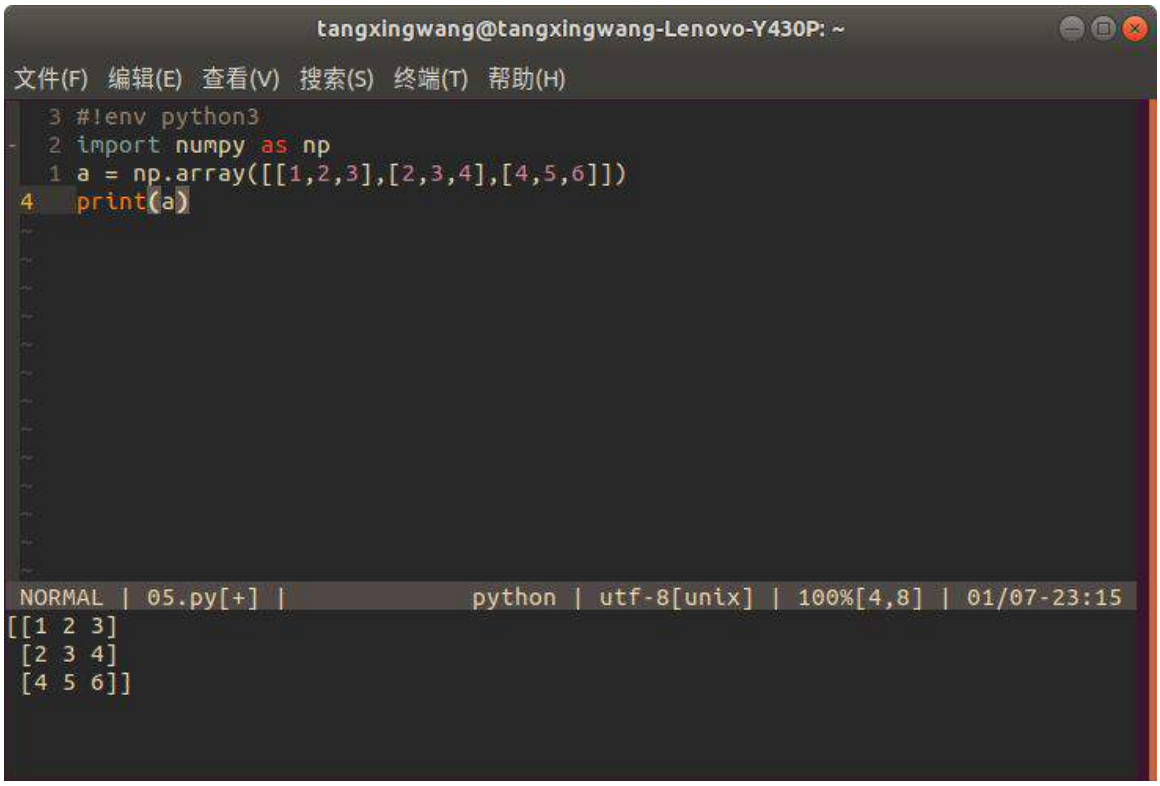
要想对数组进行运算操作，我们必须先创建个数组。方法如下：

```
import numpy as np#导入 numpy 这个包
a0 = np.array([1,2,3,4])#采用列表方式
a1 = np.array((1,2,3,4))#采用元组方式
```



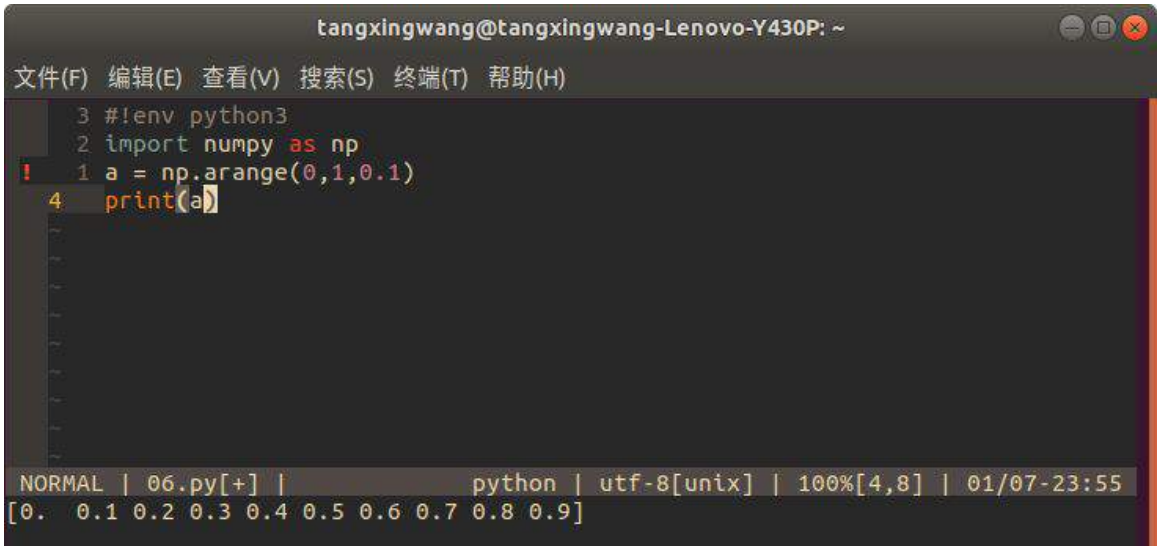
对于多维数组的创建（注意中括号），如下：

```
import numpy as np
a = np.array([[1,2,3],[2,3,4],[4,5,6]])
```



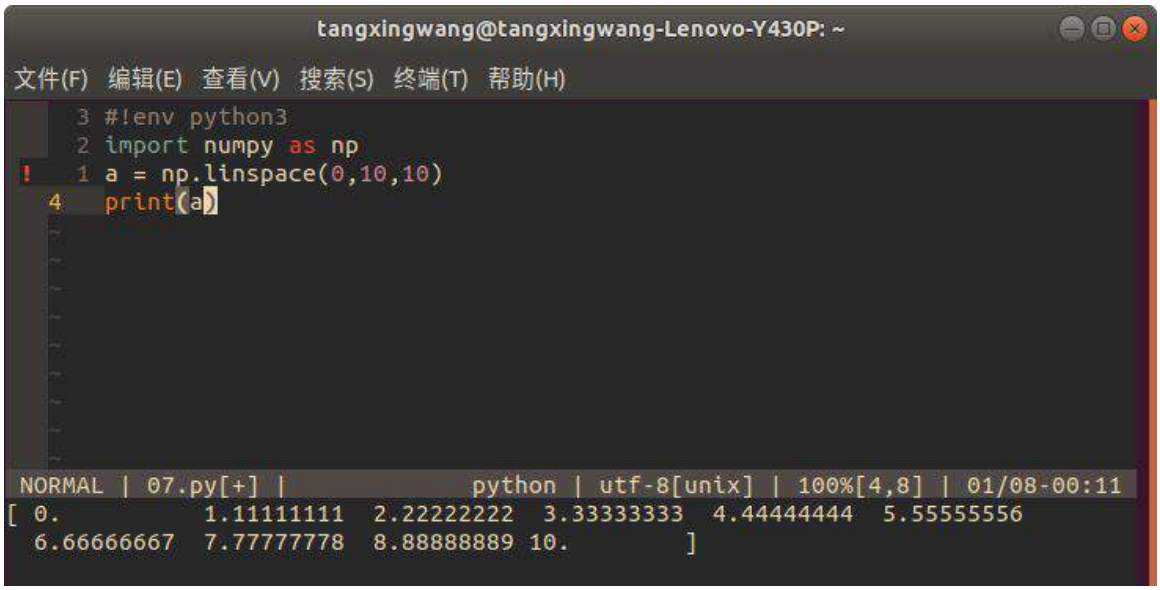
上面我们创建的数组里面的元素都是我们指定的，那么如何自动生成数组?又如何随机的生成一个数组呢？我们首先看第一个方法 arange()，如下：

```
import numpy as np
a = np.arange(0,1,0.1)
```



在上面这个数组中，arange()的第一个值代表开始值，第二个值代表终值（不包括这个值），最后一个值代表步长（间隔），如 arange(1,10,1)代表一个从 0-9，步长为 1 的数组。这就是 arange()，经常用的到！我们再看第二个方法 linspace()，如下：

```
import numpy as np
a = np.linspace(0,10,10)
```



对于 linspace()，它的前两个值和 arange()一样，代表开始值和终值，但有个区别是 linspace()默认包括终值，如果你不想包括终值，加上 endpoint = False 即可，对于第三个值它是指元素的个数，这个和 arange 不一样，一定不要混淆。

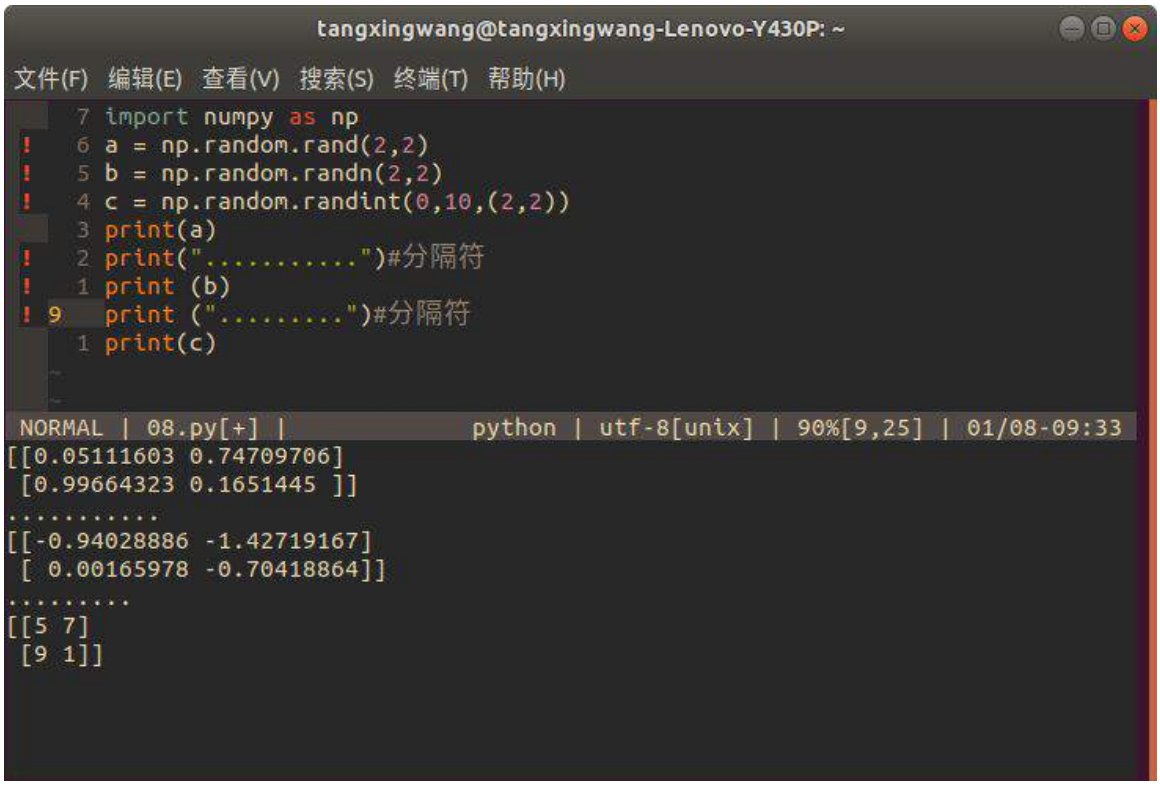
最后我们再说下如何创建一个随机数组。

在 NumPy 中有一庞大的函数库，对于随机数我们可以采用 numpy.random 模块，该模块中有大量和随机数相关的函数。一些函数如下：

函数名	功能	函数名	功能
rand	产生 0 到 1 之间的随机数	uniform	产生符合均匀分布的随机数
randn	产生符合标准正态分布的随机数	seed	产生随机种子数
randint	产生指定区间的随机整数	choice	从指定的样本中随机选择数据
normal	产生符合正态分布的随机数	shuffle	将指定的样本的元素顺序打乱

我们可以利用这些函数来创建你想要的随机数，一些实例如下：

```
import numpy as np
a = np.random.rand(2,2)
b = np.random.randn(2,2)
c = np.random.randint(0,9,(2,2))
```



创建随机数是不是很简单，其实对于数组的创建还有许多方法,如下面所示：

```
np.zeros() :生成元素全是 0 的数组
np.ones()：生成元素全是 1 的数组
np.zeros_like(a):生成形状和 a 一样且元素全是 0 的数组
np.ones_like(a):生成形状和 a 一样且元素全是 1 的数组
...
```

相信通过上面的介绍你已经掌握了如何创建一个数组了，很好！那么我们再思考一个问题，若碰到一个元素很多的数组，但却不知道它的形状等参数，这时该怎么办呢？对于这个问题我们可以通过下面的一些方法来解决。

```
获取数组 a 的 shape:a.shape
获取数组 a 的元素类型：a.dtype
获取数组 a 的维度：a.ndim
....
```

2.3 存取数组

当一个数组创建好后，我们有时候可能需要对一个数组中的一些具体元素进行运算，或者更改数组中一些元素的值。进行这些操作的前提是先能存取数组，为了解决这个问题，这里我们主要介绍切片法和整数列表来存取数组元素，这种方法其实也是最常见的。

```
import numpy as np
a = np.array([4,2,3,5,9,0,,6,8,7])
```

```
>>>[4,2,3,5,9,0,6,8,7]
```

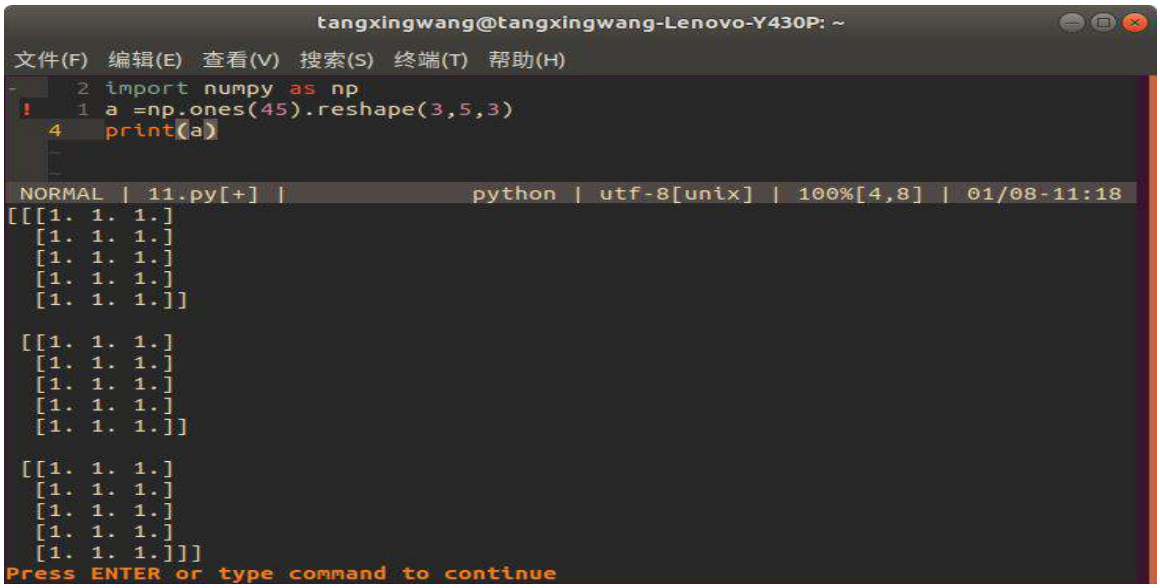
对于上面这个数组，如果我想要得到 5 这个元素该怎么办呢？很简单，在 ndarray 中第一个元素的位置是 0，本例中 5 在第四个位置，所以 a[3] = 5。

我们还可以用切片获取数组的一部分，如 a[3:5]表示获取第四个位置和第五个位置的元素,a[3:5]=[5,9];a[::-1]表示从最后一个元素到第一个元素，该方法省略了开始下标和结束下标，这时候开始下标就是对应第一个元素，结束下标就对应着最后一个数，-1 表示步长为 1，负号从后往前数。

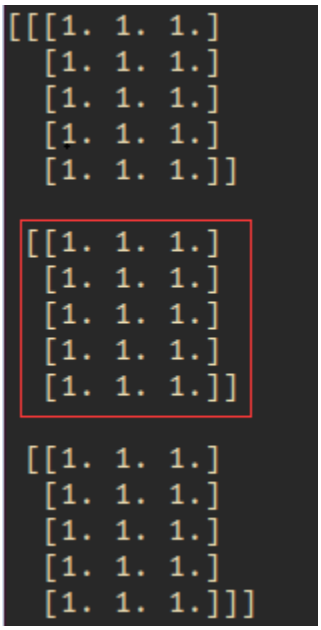
上面说的都是一维数组的存取，我们再来说一下二维数组。其实二维数组和一维数组数组类似，只是二维数组有 2 个轴，所以下标自然需要 2 个值来表示。请看下面的实例：

在二维数组中竖轴表示第 0 轴，横轴表示第 1 轴，读取元素时我们通过逗号把 0 轴和 1 轴隔开，这样就可以通过一维数组的方法来读取，最后两者的交集就是我们需要读取的元素。

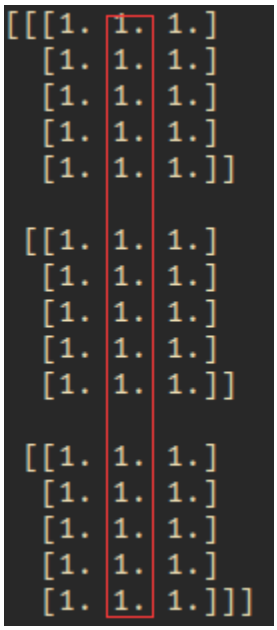
我们再看下三维数组，这也是最复杂的，在深度学习特征数据处理时用的是最多的。我们先创建一个 3 行 5 列 3 通道的数组，看看效果。



再来分析下这个生成的数组。我们知道这个三维数组有下图所示的三块，而第几块又代表通道的第几行数据，图中圈的那个块就是通道的第 2 行数据，另外在每一个块里面每行数据代表通道的第几列数据。图中圈的那个块 5 有行数据，则代表着这个通道有 5 列数据。

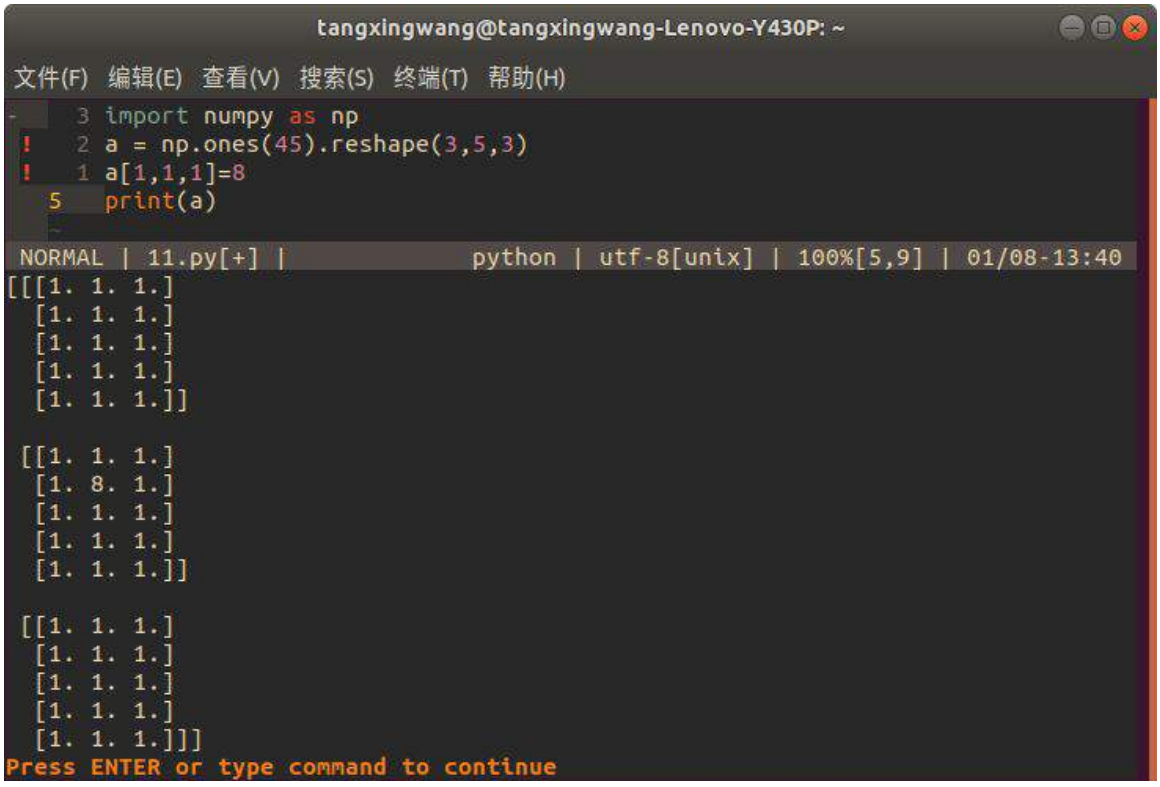


其次在这个三维数组中，有下面图示的这样三列，一列代表一个通道。另外要注意所有的数据位置的下标都是从 0 开始。



下面我要把图示的元素改成 8 该怎么办呢？如下：

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



通过上面的例子你是否理解了三维数组应该怎样存取数据了呢？理解了的话就打开 vim 多写写基本就能深刻的掌握了。

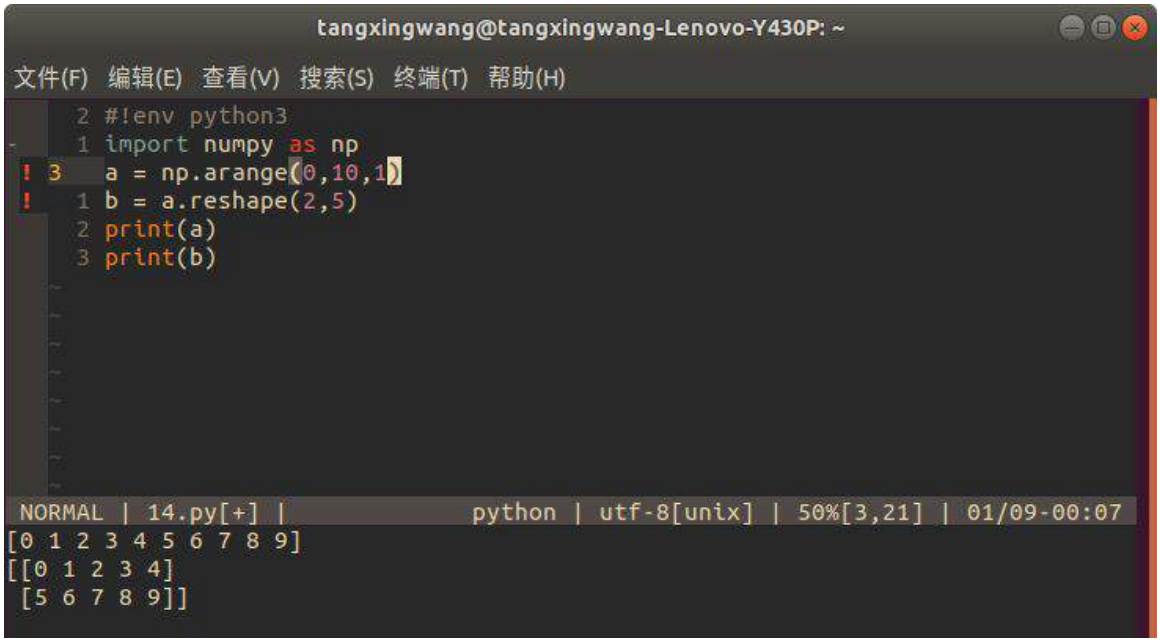
2.4 NumPy 常见函数使用

现在我们已经学会了创建数组和数组的存取，那么我们该如何对数组进行函数运算呢，这也是 NumPy 的核心内容。

2.4.1 数组维度变换

我们首先说一下如何对数组的形状进行整理，即将一个任意形状的矩阵转化我们想要转化的任意形状，当然要想完成这个操作，元素个数必须要满足。请看下面实例：

```
import numpy as np
a = np.arange(0,10,1)
b = a.reshape(2,5)
print(a)
print(b)
```



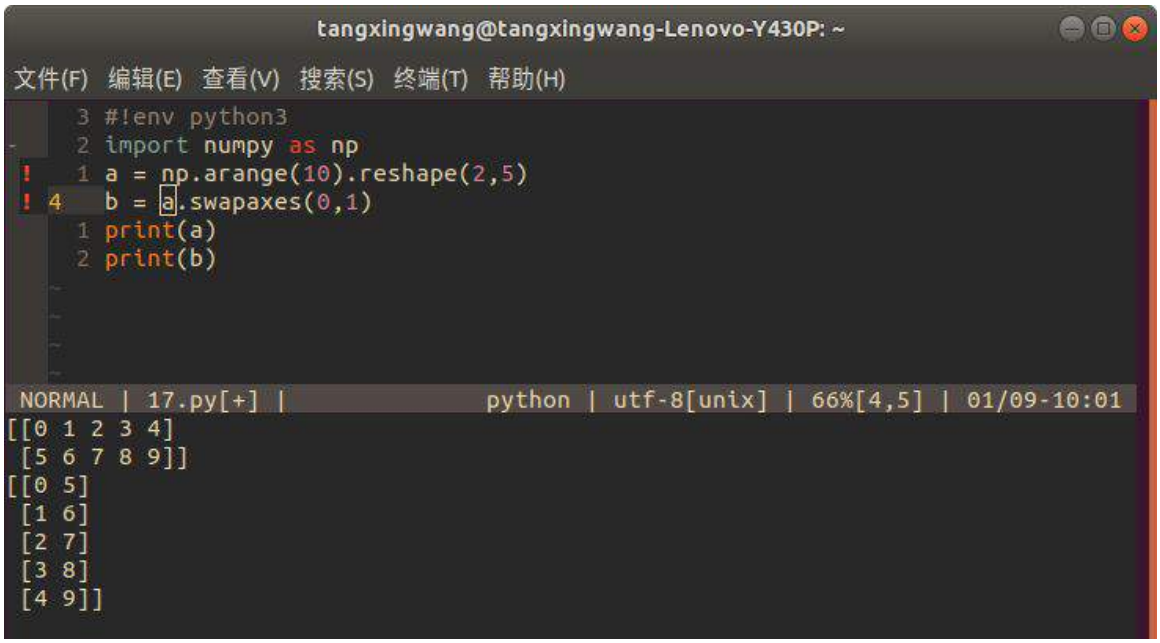
上面的实例通过 reshape()函数把一个 1 维数组，变成了一个 2 行 5 列的一个数组。reshape()里面的参数就是你想要转换成的数组的形状。再看一个实例对 reshape()熟练下，如下：

```
import numpy as np
a = np.arange(0,10,1)
b = a.reshape(2,-1)
c = a.reshape(-1,5)
print(a)
print(b)
print(c)
```

b 和 c 的结果是一样的，而且和上一个实例的结果也一样，这是为什么呢？其实这里面的-1 代表自动生成的意思，意思就是对于 b 我已经指定了数组的行是 2 行，那么系统就会自动生成一个 5 列，因为是 10 个数，必须是 5 列，所以 b 和 c 仍然是 2 行 5 列的数组，这就是数组形状变换。

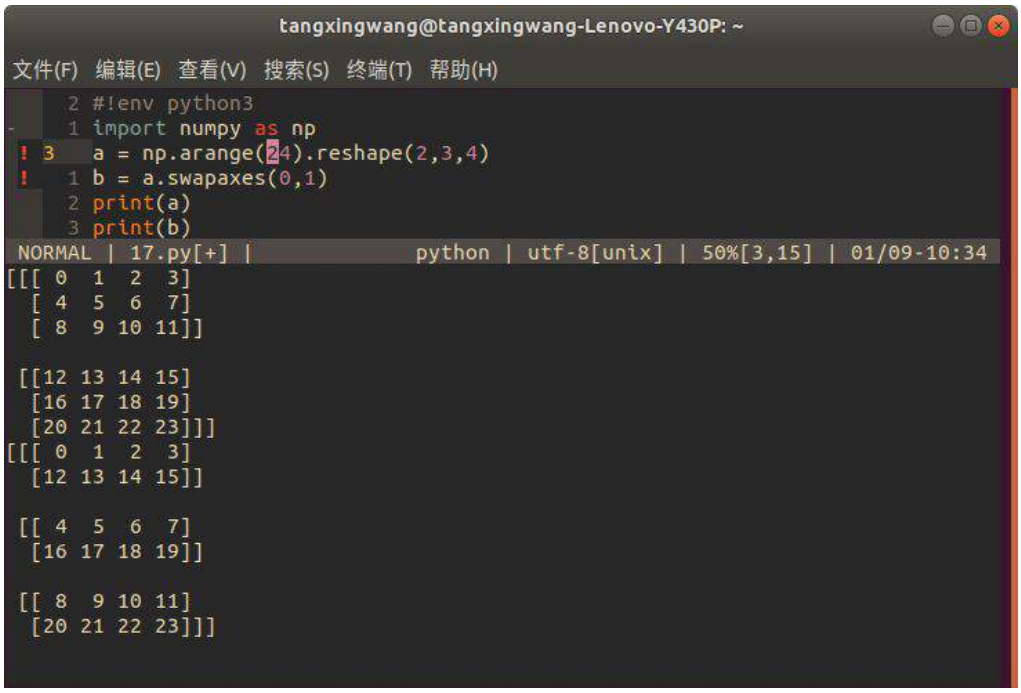
说完数组形状变换我们再看下如何对数组进行维度交换。请看下面实例：

```
import numpy as np
a = np.arange(10).reshape(2,5)
b = a.swapaxes(0,1)
print(a)
print(b)
```



通过上面实例我们看出通过 swapaxes()将一个数组的第 0 轴和第 1 轴进行了交换，由 2 行 5 列变成了 5 列 2 行。这是二维数组的维度交换，我们再看一个三维数组的例子，如下：

```
import numpy as np
a = np.arange(24).reshape(2,3,4)
b = a.swapaxes(0,1)
print(a)
print(b)
```

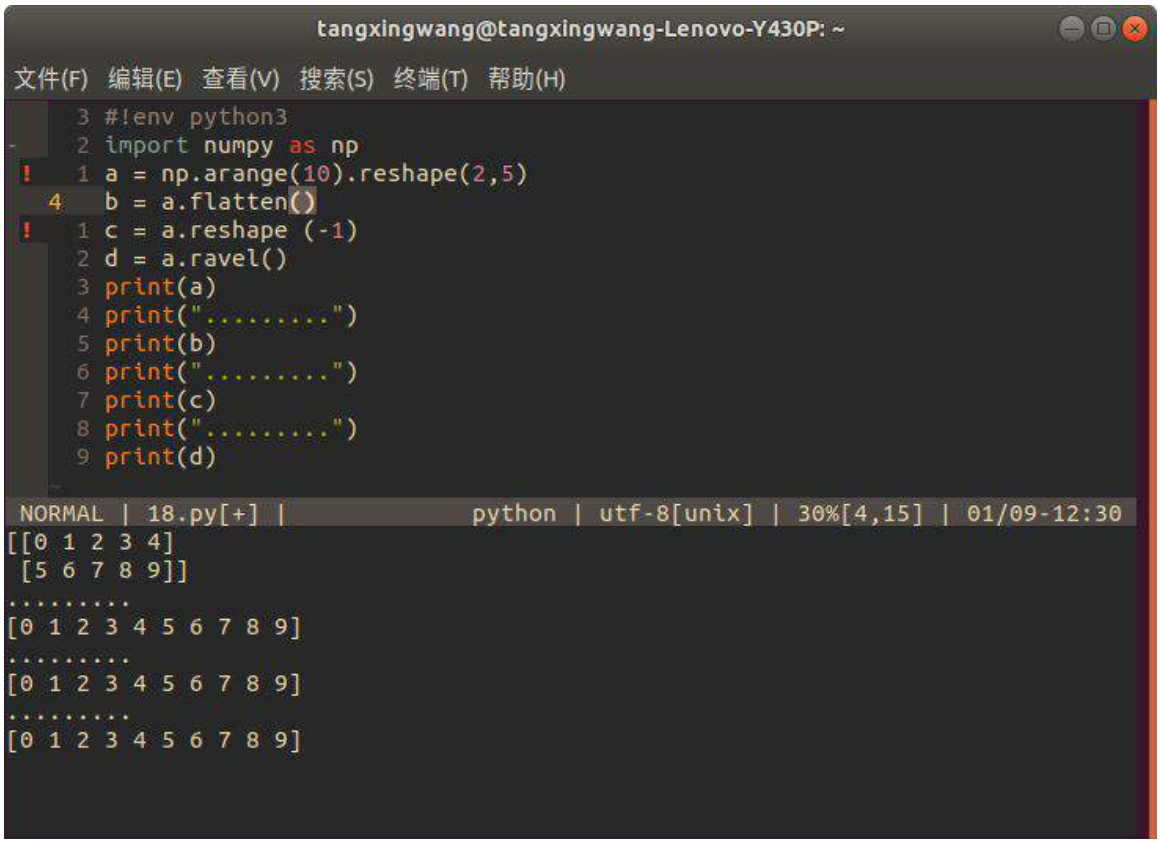


这个实例我将三维数组的第 0 轴和第 1 轴进行了交换，第 0 轴就是我在上面 2.3 存取数组这一节中说的块，第 1 轴就是块中的行，下面我将我对三维数组维度交换的理解和大家分享下。

如下，我们首先将第一块第一行的[0,1,2,3]的位置记为(1,1)，第一块第二行的[4,5,6,7]的位置记为(1,2)，第二块第三行的[20,21,22,23]记为(2,3)，其它几个位置坐标类推。现在我们需要将第 0 轴和第 1 轴交换，所以第一块第一行的[0,1,2,3]的位置变为(1,1),就是第一块第一行；第一块第二行的[4,5,6,7]的位置变为为(2,1),就是第二块第一行；第二块第三行的[20,21,22,23]变为(3,2),就是第三块第二行。通过这样的理解你对上面实例输出的结果明白了吗？明白的话，请继续往下学如何对数组进行降维。

对于数组降维，我们继续通过实例来分析，如下：

```
import numpy as np
a = np.arange(10).reshape(2,5)
b =np.flatten()
c = a.reshape(-1)
d = a.ravel()
print(a)
print(b)
print(c)
print(d)
```

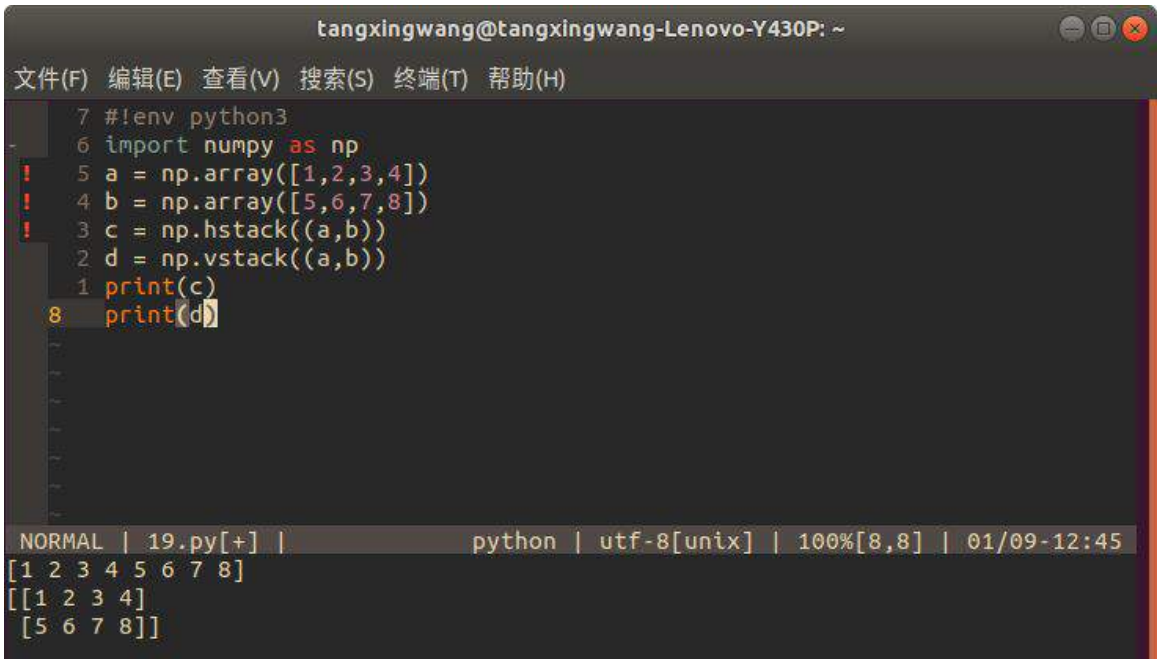


可以看出我们通过 reshape(-1)、flatten()和 ravel()函数将多维很容易就变成了 1 维数组。

2.4.2 堆叠数组

我们再说一下数组的堆叠，这个也是经常会用的。数组的堆叠通常有水平叠加和垂直叠加，分别用到 hstack()和 vstack()函数，请看下面的实例：

```
import numpy as np
a = np.array([1,2,3,4])
b = np.array([5,6,7,8])
c = np.hstack((a,b))
d = np.vstack((a,b))
print(c)
print(d)
```



通过这个例子我们也看出通过 hstack()和 vstack()将数组 a 和 b 堆叠成了一个数组。

上面就是我对 NumPy 在深度学习中最常见的几点的介绍，其实还有许多，平时多多积累就行。

3 数据可视化——matplotlib

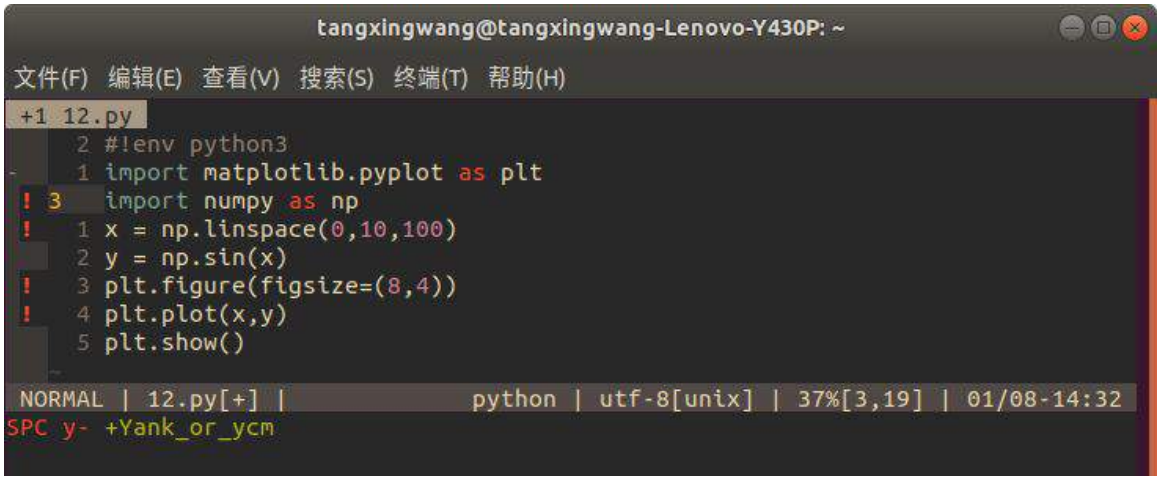
说完 python 我们再说说深度学习中用的比较多的 matplotlib。matplotlib 是 python 中最常用的可视化工具之一，用处非常大。

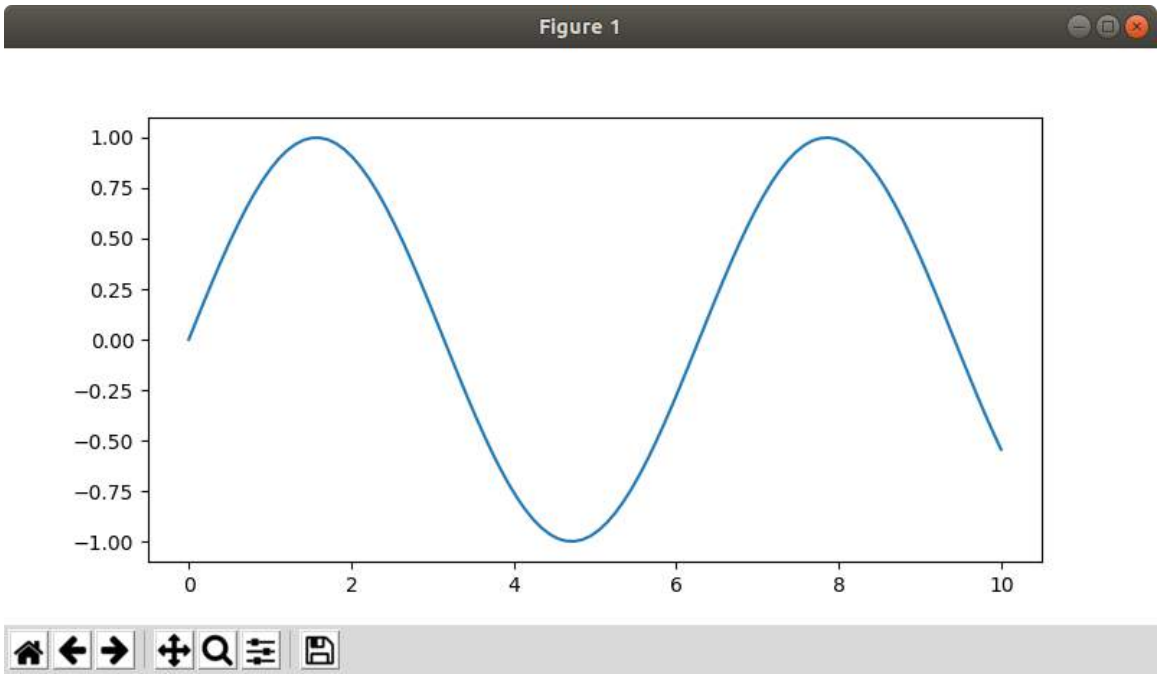
3.1 使用 pyplot 模块绘图

我们先通过 matplotlib 和 NumPy 绘制一个图像。

```
import matplotlib.pyplot as plt
```

```
import numpy as np
x=  np.linspace(0,10,100)
y=np.sin(x)
plt.figure(figsize=(8,4))
plt.plot(x,y)
plt.show()
```





在这个实例中，我们首先通过 import matplotlib 的绘图块 pyplot，并重新命名为 plt。然后用 figure 调出一个画布,figsize 参数指定画布的宽度和高度，单位是英寸（1 英寸为 25.4 毫米）。创建好画布后，我们就可以用 plot()在画布上画图了，plot()

的前两个参数分别代表 X，Y 轴数据的对象。另外 plot()参数还可以指定曲线的标签，颜色，线宽等。

其实我们还能对坐标轴通过下面的方法进行一些参数的设置：

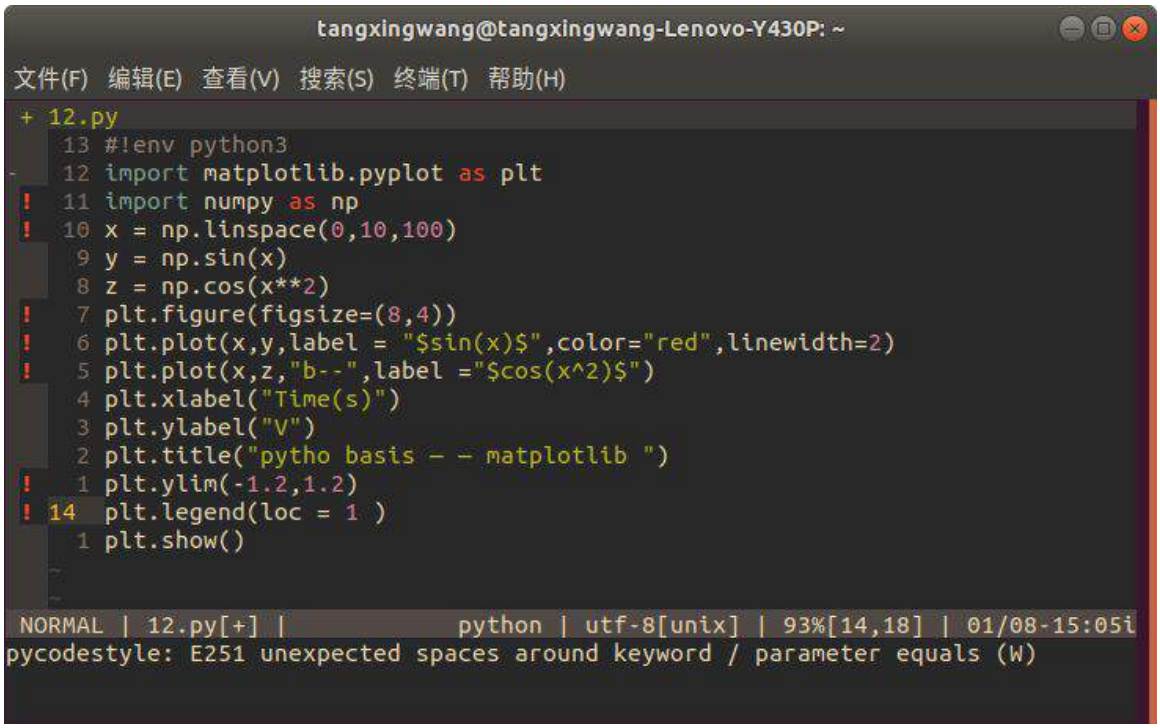
xlabel,ylabel：分别设置 X，Y 轴的标题文字

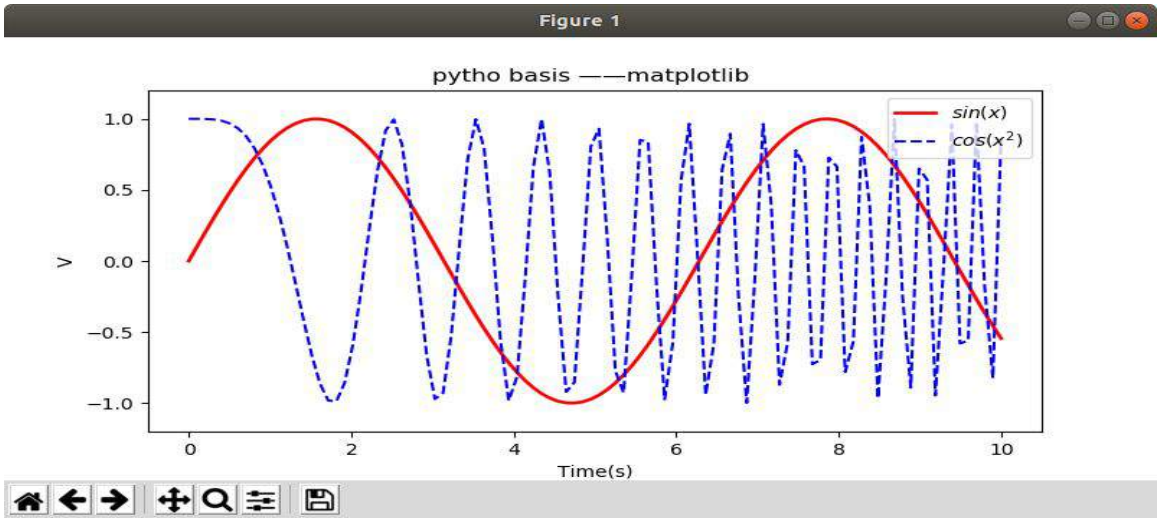
title：设置标题

xlim,ylim：分别设置 X,Y 轴的显示范围

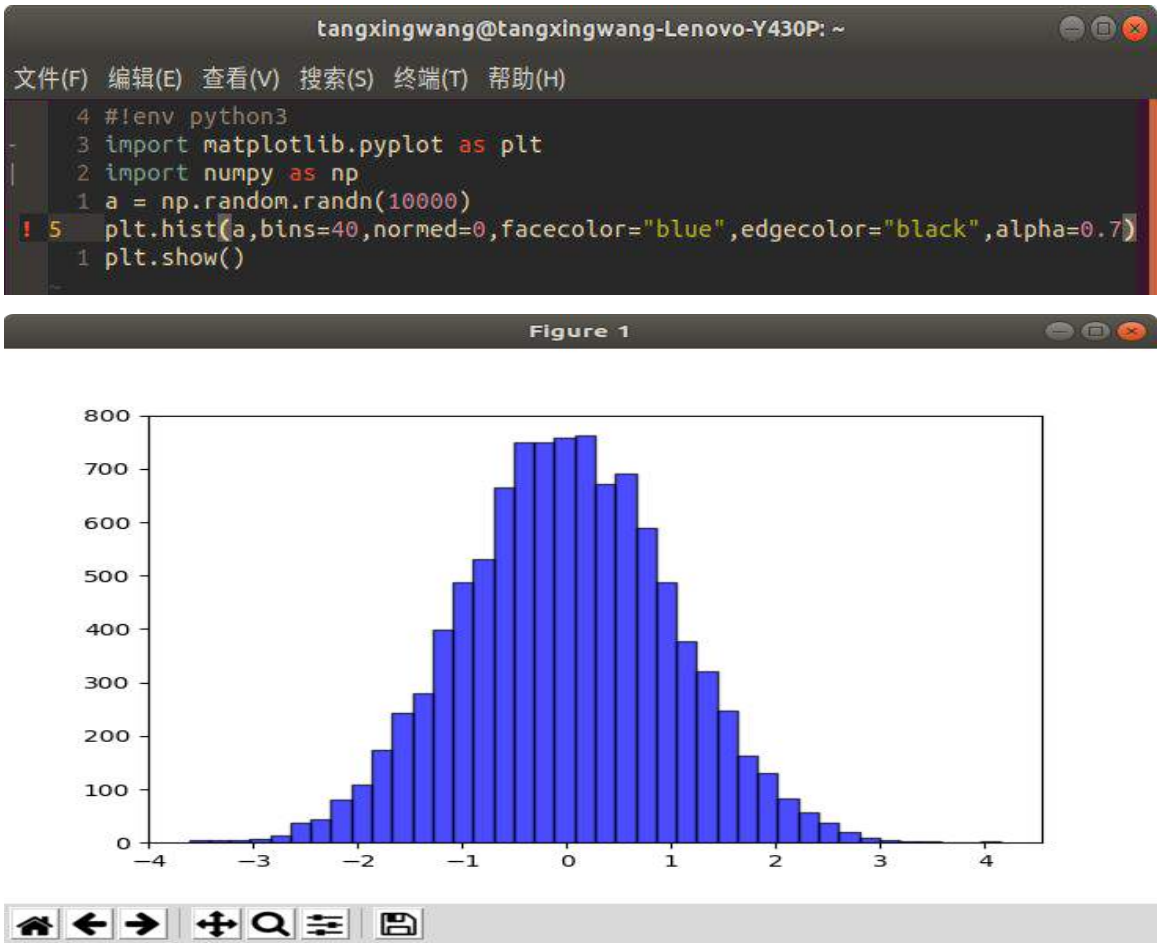
legend:显示图例

请看下面一个标准的图形：





接下来我们再看看如何画直方图，直方图在图像处理中经常会用到。



在用 plt.hist()画直方图时，第一个参数是绘图数据，这是必须要有的；另外 bins 代表直方图的长条形数目，默认为 10；normed 表示是否将得到的直方图向量归一化，默认为 0，代表不归一；facecolor 代表长条形的颜色；edgecolor 代表长条行边框的颜色；alpha 代表透明度。

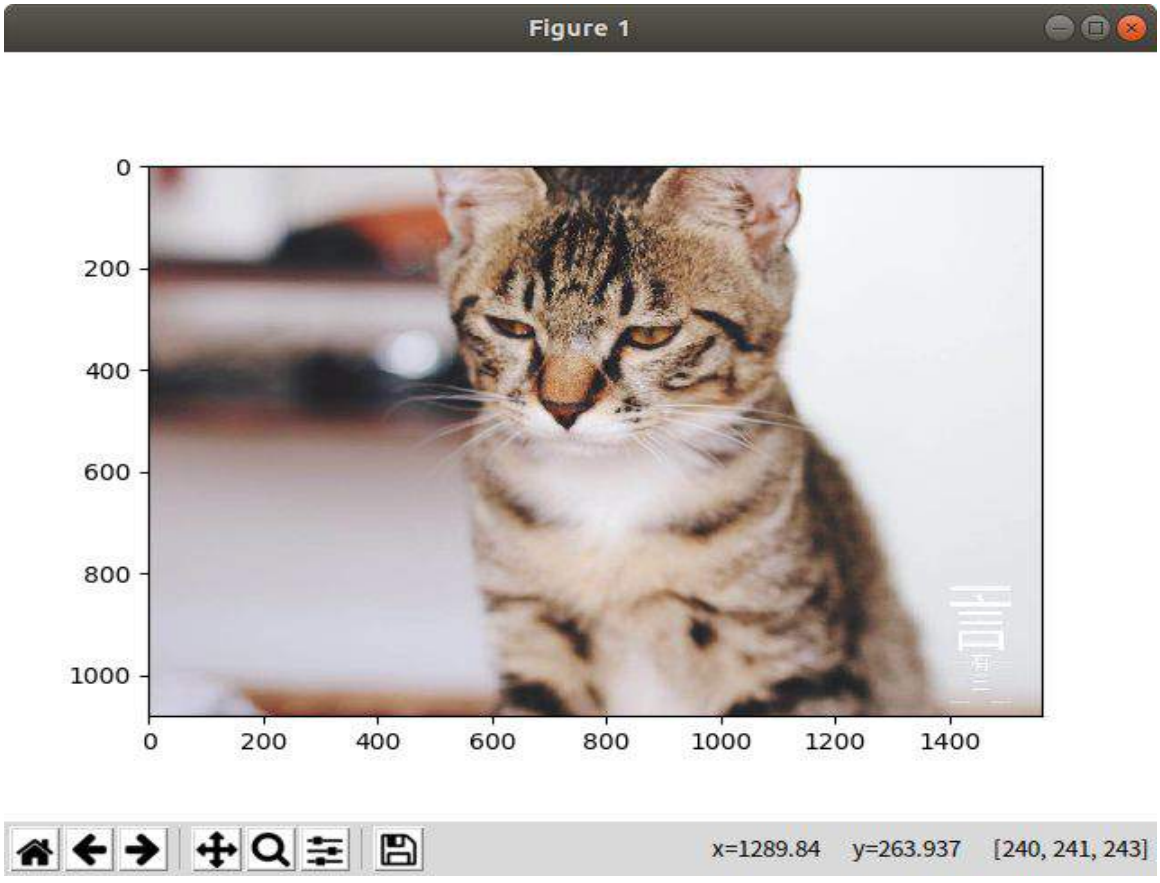
3.2 matlibplot 读取图像

matplotlib 的 imread 和 imshow()提供了图像的读取和显示功能，另外 imread()从图像文件中读入数据得到的是一个图像的 NumPy 数组。

现在我们用 matlibplot 读取上面这一张可爱的猫图，方法如下：

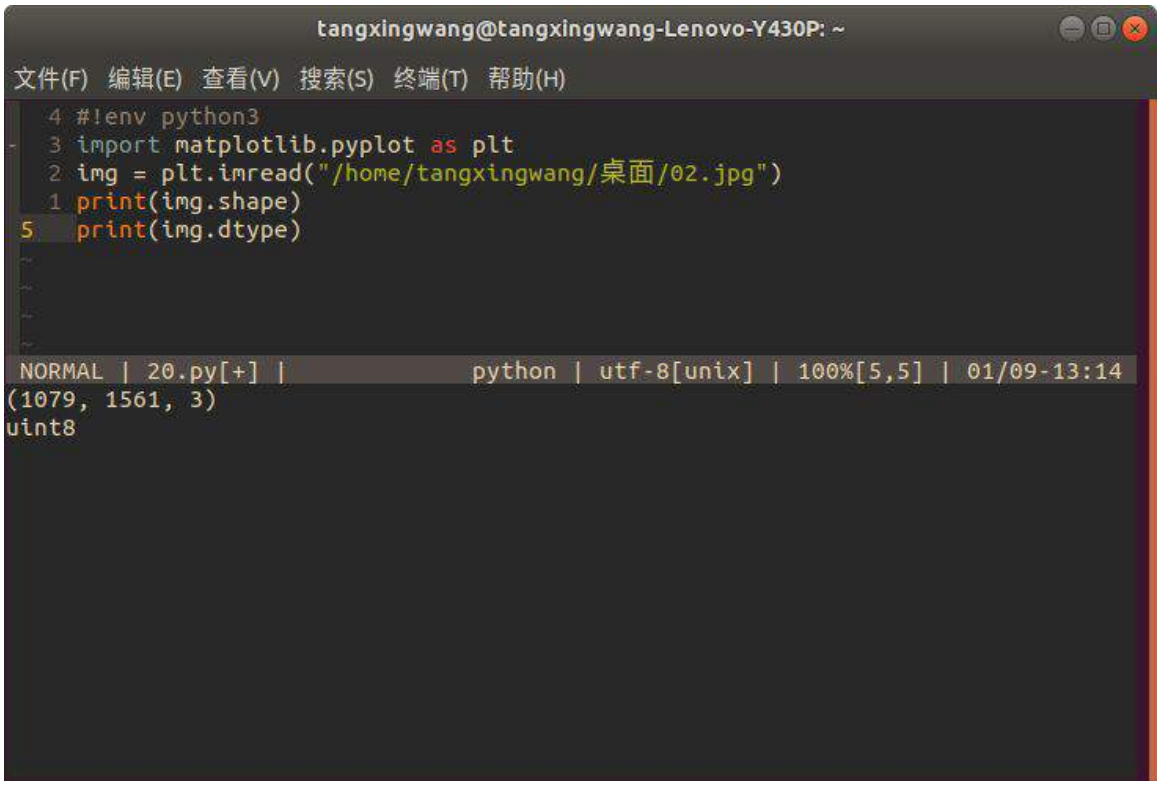
```
import matplotlib.pyplot as plt
img = plt.imread("02.jpg")
plt.imshow(img)
plt.show()
```

```
tangxingwang@tangxingwang-Lenovo-Y430P: ~
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
  4 #!env python3
  3 import matplotlib.pyplot as plt
  2 img = plt.imread("/home/tangxingwang/桌面/02.jpg")
  1 plt.imshow(img)
5   plt.show()
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
 NORMAL | 20.py |                python | utf-8[unix] | 100%[5,1] | 01/09-13:10
"20.py" 5L, 126C
```

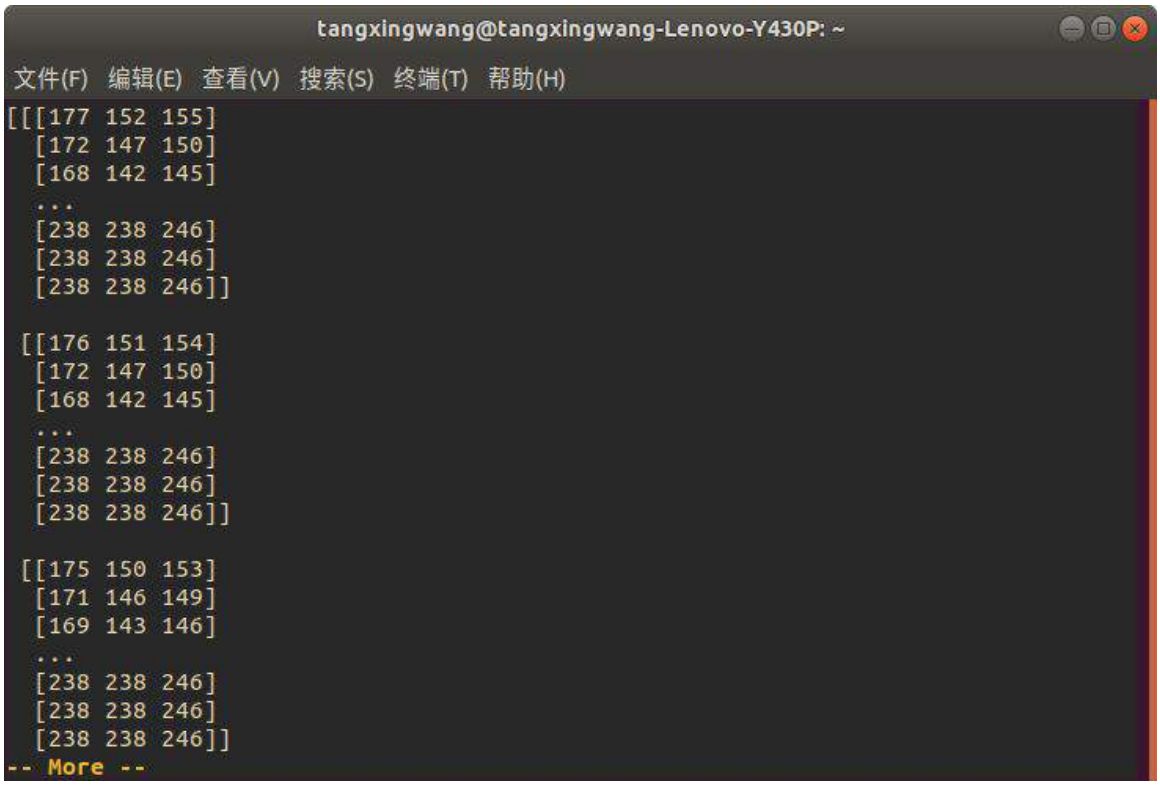


可以看出，很容易就能读取一张图片。我们可以通过下面一些方法查看这张图片的属性。

```
print(img.shape)
print(img.dtype)
```



我们再看看 matplotlib 读取的图像是不是 NumPy 数组，如下：



可以明显看到这是 ndarray 格式，总共三个通道，分别代表 RGB。

3.3 matplotlib 工具栏

从上面的例子中就可以看到，当显示一张图片时，菜单栏自动生成了一些按钮，这些按钮都有各自的功能。

3.3.1 前进后退按钮

这三个按钮就像是我们使用的浏览器中的主页和前进后退按钮一样，一开始这三个图是没有什么用的，因为它本来就处于主页，既不能前进也不能后退，当你使用平移和缩放功能后，每一次操作就相当于在浏览器中点开了一个网页一样，这时候你就可以使用前进后退和回到最开始状态的按钮了。

3.3.2 平移缩放按钮

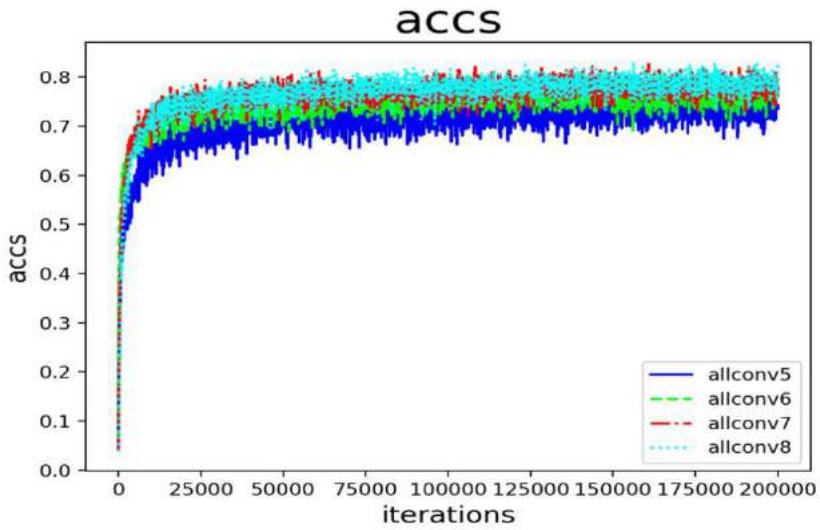
这个按钮也比较简单，按住鼠标左键在图片区域左右移动可以实现图像的左右平移，上下移动就可以使图像上下平移，按住 X 或者 Y 键移动即只能在 X 或者 Y 方向

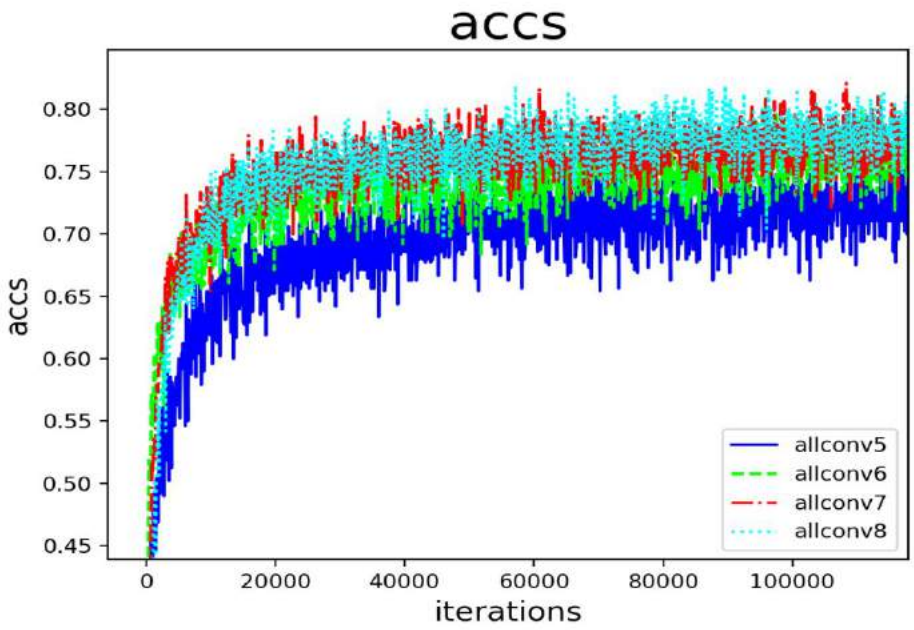
上平移。同理按住鼠标右键就是缩放。如果按住 Ctrl 键再进行上述操作，则是 XY 轴成比例平移或缩放。

3.3.3 缩放到指定矩形按钮

按住鼠标左键或者右键，选定一个矩形区域，即可将图形放大或者缩小到制定的矩形区域中。

在放大局部细节图时经常使用。





3.3.4 设置子图参数按钮

点击该按钮可以设置子绘图区域的长度和宽度，还可以设置各个子图之间的距离。

3.3.5 保存按钮

该按钮可以将图像保存为 png、pdf 等格式。

matplotlib 的一些基础知识就介绍到这里，大家一定要多加实践，这样才能更好的掌握。

4 必备的 python 库

要想深度学习学的好，必须非常好的掌握 python 各种库，这是进行深度学习的基础。下面我和大家分享下一些常用的库，其实上面我已经介绍了两个了那就是 NumPy 和 matplotlib，还有一些其他的库，请继续往下看。

4.1 科学计算与数据处理库

说到科学计算和数据处理，我们可能马上就会想到 NumPy 但其实还有两个其他的库，那就是 SciPy 和 Pandas。

Scipy 在 Numpy 的基础上增加了众多的数学计算、科学计算及工程计算中常用的模块，例如线性代数、图像处理等。在 Scipy 中的 ndimage 子模块就是致力于进行图像处理的。

Pandas 是基于 NumPy 开发，提供了众多更高级的数据处理工能。Pandas 中包含许多用于分组，过滤和组合数据的内置方法，以及时间序列功能。

有机会我们再说说这两个库。

4.2 深度学习框架

目前深度学习框架已经呈百家争鸣之态势，有 Caffe、TensorFlow、Pytorch、Keras 等等，对于我们来说，不可能都能掌握，但市面上主流的框架我们还是必须要熟练的掌握。

这些深度学习框架，我们在公众号的往期文章已经说过大半，大家可以自行翻阅。

4.3 服务端 Flask

Flask 是一个微型的服务端框架，它旨在保持核心的简单，但同时又易于扩展。默认情况下，Flask 不包含数据库抽象层、表单验证，或是其他任何已有多种库可以胜任的功能。然而，Flask 支持用扩展来给应用添加这些功能。众多的扩展提供了数据库集成、表单验证、上传处理、各种各样的开放认证技术等功能。Flask 的这些特性，使得它在 Web 开发方面变得非常流行。

4.4 前端 urllib 等

urllib 是 Python 自带的标准库，无需安装，直接可以用。提供了如下功能：网页请求、响应获取、代理和 cookie 设置、异常处理和 URL 解析，主要用于前端爬虫，对于获取和整理数据是必备的。

5 总结

AI 白身境第三讲结束了，但学习的路永无止境，python、NumPy、matplotlib、深度学习框架的知识还有很多，需要我们不断的学习。期待我们的下期吧。

【AI 白身境】深度学习必备图像基础

本篇是**《AI 白身境》**的第四篇，所谓白身，就是什么都不会，还没有进入角色。

我们已经说了 linux 基础和 python 基础，接下来就要开始真正干活了。所谓万丈高楼平地起，正式从事深度学习技术的三大方向，图像，语音，NLP 之前，自然要先了解各自的基础。笔者身处计算机视觉领域，所以这一期就跟大家说说**必备的图像基础**。

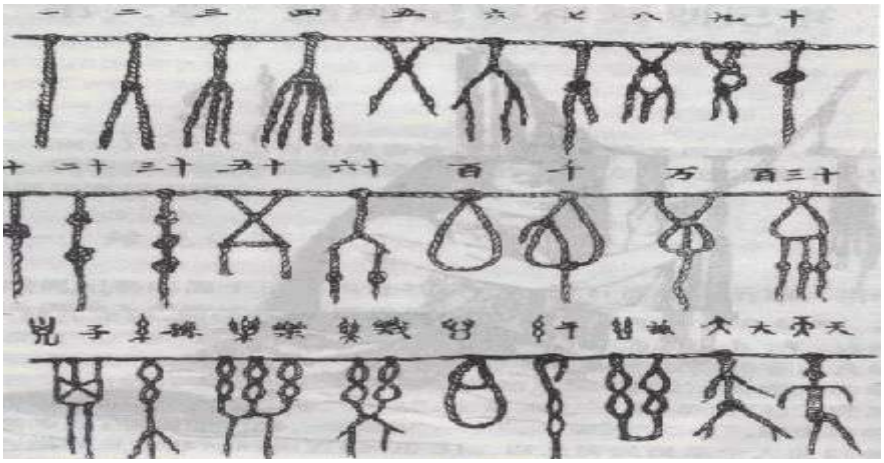
作者|言有三

1 图像的起源

1.1 图像的进化

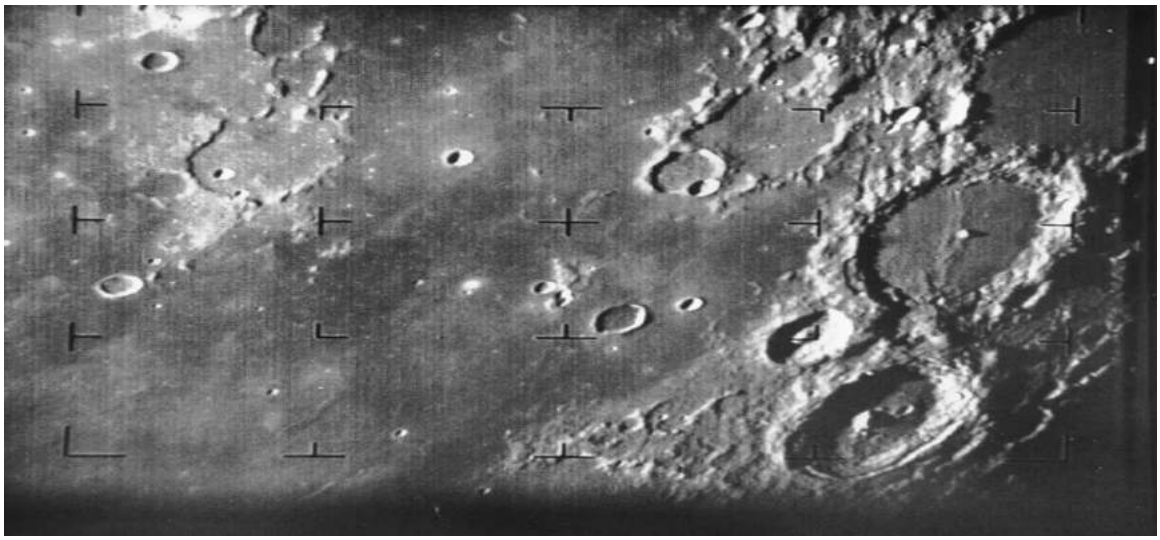
图像是什么？这个问题大家都有自己的答案。我的答案是，**图像是一门语言**，是人类文明的象征。

人类起源时没有图像，最开始记事采用的方法是什么呢？据《易·系辞下》中记录，“上古结绳而治，后世圣人易之书契 ，百官以治，万民以查”，也就是说，最开始没有文字，大家采用结绳的方法来记录。



到了后来，中国进入了一个文明时代，商朝，并且有了自己的文字，甲骨文，这就是我们现在汉字的起源。

再后来，随着西方文明的发展，有了照片，从此我们进入了多媒体记录信息的时代。



如今，图片视频已经成为了人的一生中非常重要的记忆载体。

英文 image 来源于拉丁文 imāgō，它的含义有很多，比如 reflection，visible form 等等，实际上表述的就是一种语言。

图像包括图和像，图，它是一直客观存在的光的分布。而像则是图在人大脑中的印象。

1.2 模拟图像与数字图像

图像起源于 1826 年前后法国科学家 Joseph Nicéphore Niépce 发明的第一张可永久保存的照片，属于**模拟图像**。

模拟图像又称连续图像，它通过某种物理量（如光、电等）的强弱变化来记录图像亮度信息，所以是连续变换的。模拟信号的特点是容易受干扰，如今已经基本全面被数字图像替代。

在第一次世界大战后，1921 年美国科学家发明了 Bartlane System，并从**伦敦传到纽约传输了第一幅数字图像**，其亮度用离散数值表示。

这是一种电缆图片传输系统，将图片编码成 **5 个灰度级**，1929 年发展成 15 个灰度级，通过海底电缆进行传输。在发送端图片被编码并使用打孔带记录，通过系统传输后在接收方使用特殊的打印机恢复成图像。

二战时,世界各国报纸上的图像都是采用 Bartlane System 进行传输。

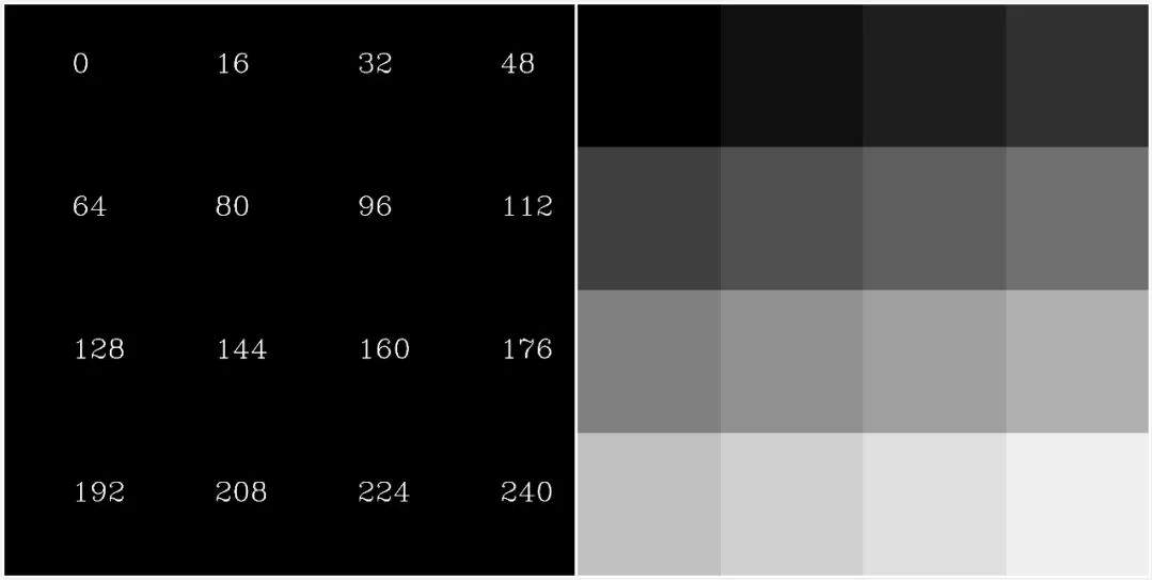
1950 年左右，计算机被发明，数字图像处理学科正式诞生。

模拟图像和数字图像的对比，大家可以看看。

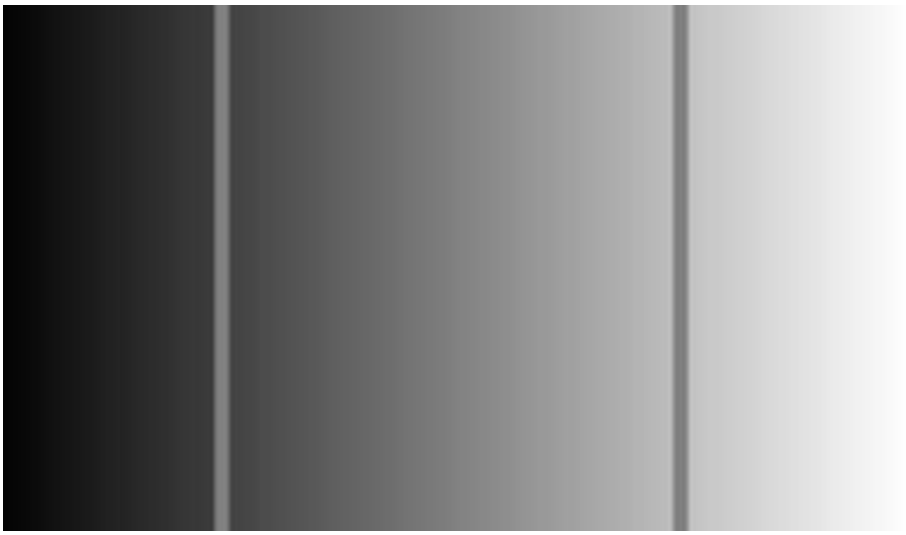
2 数字图像表示

2.1 位数

计算机采用 0/1 编码的系统，数字图像也是利用 0/1 来记录信息，我们平常接触的图像都是 8 位数图像，包含 0～255 灰度，其中 0，代表最黑，1，表示最白。



其实人眼对亮度的对比的敏感度远远超过亮度的本身。



就像上面的两条线，是一样的灰度值，但是人眼很难分辨这是相同的灰度，尽管知识告诉我们它是。实际上，人眼能分辨的灰度级不到 32 级，大于 16 级。

2.2 分辨率

数字图像有两个分辨率，**图像分辨率与输出分辨率**。

图像分辨率指的是每英寸的像素数，简写为 ppi，我们平常说一张图片大小的时候使用的就是图像分辨率。

输出分辨率指的是设备输出图像时每英寸可产生的点数，简写为 dpi，这是在印刷行业，摄影行业常用的分辨率，摄影行业通常要求 DPI 不低于 300。

相同的图像分辨率，更高的 DPI 表现为物理尺寸更小。因为这个时候每英寸点更多，像素变小。

如下面两张图，左图的 DPI=72，物理尺寸大小为高 46.85 厘米，宽 67.73 厘米。

右图的 DPI=150，物理尺寸大小为高 22.47 厘米，宽 32.49 厘米。两者的像素数是相等的，都是 1920*1328 像素分辨率，但是右边的 dpi 更大。

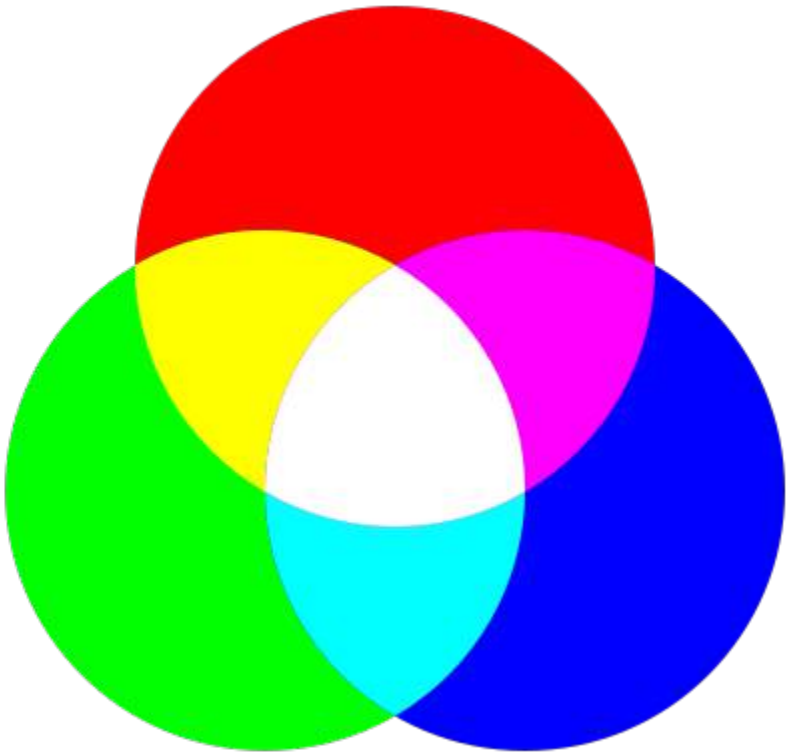

物理尺寸相同，DPI 较低表现为较低的分辨率，此时每英寸的点数变少，像素变大。如下面两张图，图像实际大小相等，但是右边的图像分辨率较低，像素数较少，清晰度有所下降。


2.3 彩色空间

图像有灰度图有彩色图，灰度图即只包含亮度信息，而彩色图不仅包含亮度信息还包含颜色信息。

我们平常接触的是 RGB 彩色图，即由红（Red）绿（Green）蓝（Blue）3 个通道组成，一张图像的每一个像素由矢量（R，G，B）表示。

这是在消费市场最广泛使用的，最常用的用途就是显示器系统，计算机、电视机等都是采用 RGB 颜色空间来进行图像显示。RGB 颜色空间背后的生物学原理是人眼有对这 3 种颜色最敏感的细胞，在自然界中肉眼所能看到的任何色彩都可以由这三种色彩叠加而成，因此也被称为加色原理。比如黄色，可以通过红色和绿色相加，全红色为（255，0，0），全绿色为（0，255，0），全黄色为（255，255，0）。



有艺术背景的读者会提出绿色和红色混合在一起产生的是褐色，与这里的计算机色彩模型加色原理不同，这是因为绘画遵循的是减色模型。

除了 RGB 颜色空间，常用的颜色空间还有 HSV，CIELab 等，我们以后会集中讲述。

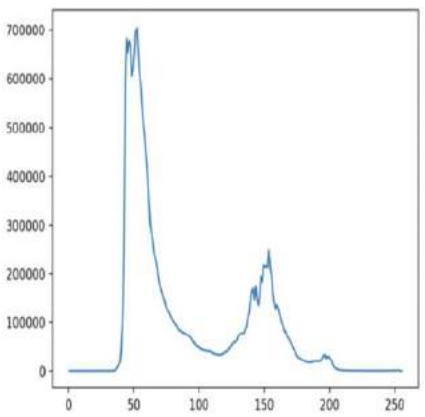
3 数字图像处理基础

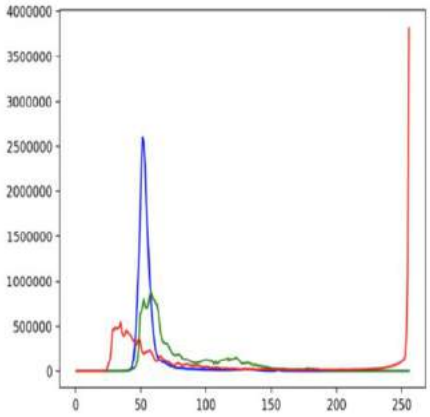
数字图像处理有一些基本的表述和概念我们必须清楚。

3.1 直方图

图像之所以能处理，是因为像素与像素是有空间联系的，对像素灰度值进行统计，就得到了直方图。

下面分别是上面灰度和彩色图的直方图。

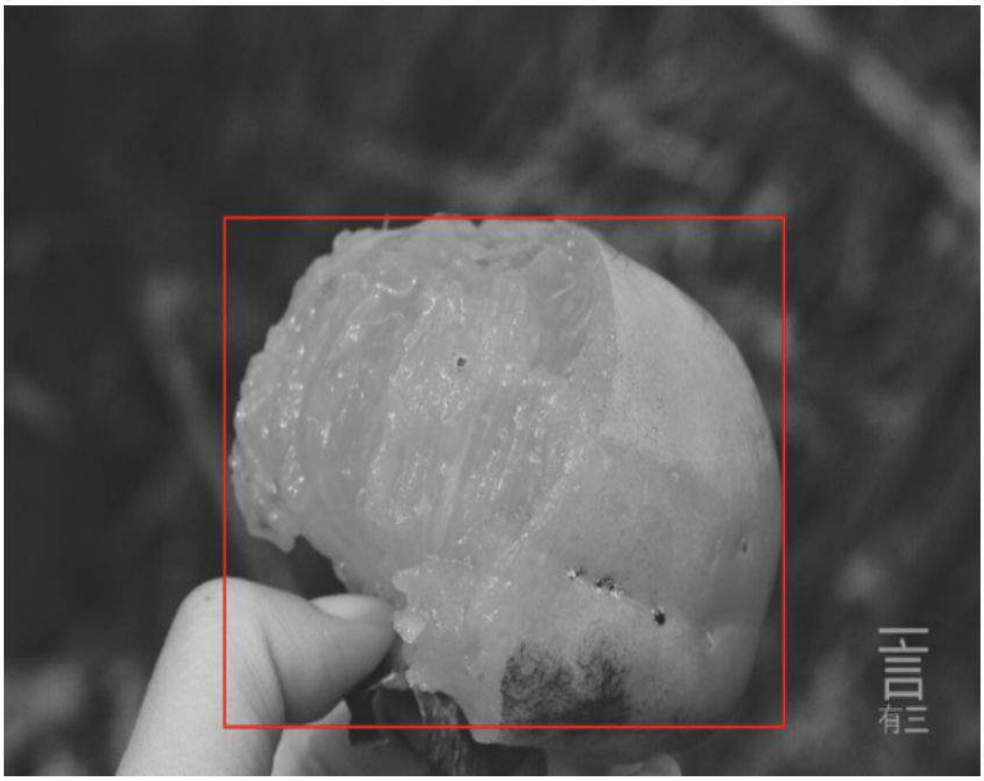




统计代码如下：

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import sys
import os

filename=sys.argv[1]
img=cv2.imread(filename)
colors=['blue','green','red']

for i in range(3):
   hist,x=np.histogram(img[:,:,i].ravel(),bins=256,range=(0,256))
plt.plot(0.5*(x[:-1]+x[1:]),hist,label=colors[i],color=colors[i])
plt.show()

imggray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
cv2.imwrite('sample_gray.jpg',imggray)

histgray,xgray=np.histogram(imggray.ravel(),bins=256,range=(0,256))
print xgray
plt.figure()
plt.plot(0.5*(xgray[:-1]+xgray[1:]),histgray)
plt.show()
```

我们可以看到，在灰度直方图包含两个很明显的分布，在彩色直方图的红色通道也包含两个很明显的分布，分别对应的就是**“前景”和“背景”**。

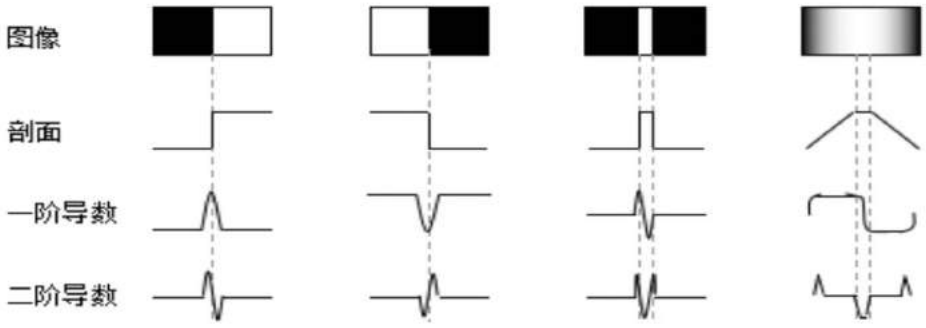


如上图，感兴趣的是图中的“柿子”，这就是前景，它的灰度比较高，对应的就是直方图中的较高峰。

3.2 边缘

视觉机制和马赫达效应都表明人眼对不连续的东西是最敏感的，而图像中不连续的东西，表现出来就是图像边缘。

边缘包含上升阶跃型、下降阶跃型、屋脊型、脉冲型等类型，



边缘检测在计算机视觉与图像处理中基础且应用广泛。通过提取目标的轮廓，用于识别不同的物体，或作为图像的特征表示。边缘检测的基本方法有很多，它们的绝大部分可以划分为两类：基于一阶导数和二阶导数的方法。

关于边缘检测方法，公众号有视频公开课，大家可以去看。

3.3 对比度与清晰度

图像有高亮度也有低亮度，对应的就是白与黑，目前多数显示系统利用 8 字节，即灰度值 0 代表最黑，灰度值 255 代表最亮，不过大部分图像上的亮度范围通常都小于最大最小值之差。

对比度，指的就是画面的明暗反差程度。

对比度有全局对比度和局部对比度。增加对比度，画面中亮的地方会更亮，暗的地方会更暗，明暗反差会增强。下面分别是降低对比度和增加对比度，感受一下。

清晰度，指的是边缘附近的敏感对比。

如果增加清晰度，边缘较暗的一侧会变得更暗，边缘较亮的一侧会变得更亮，轮廓更加清晰，不过调节过度，会出现晕影。

增加清晰度，可以通过锐化操作来进行。降低清晰度，可以通过降低图像分辨率，增加模糊等方法。

4 图像处理与计算机视觉

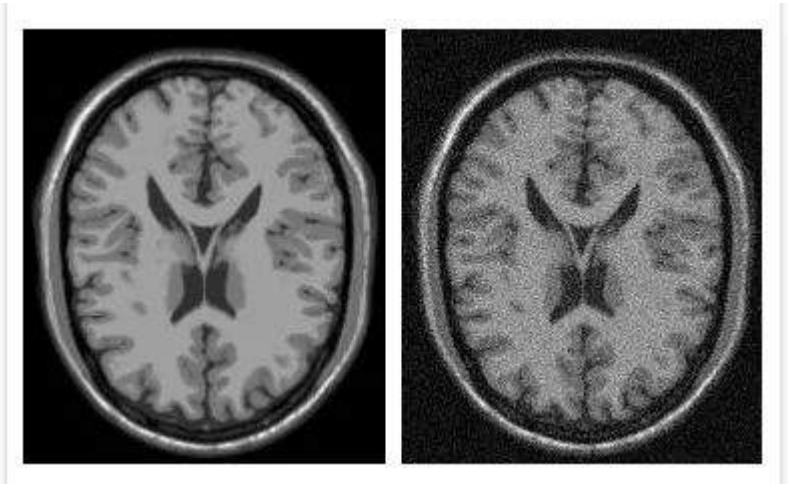
有一些基本概念容易混淆，图像处理，图形学，计算机视觉等，用几个图就很好理解了。

4.1 图像处理领域

图像处理一般指数字图像处理，输入是图像，输出也是图像，通常是为了改善，增强图像的内容以方便后续的分析。

图像模糊

对比度增强



图像降噪

更多的图像算法，等我更新即可。

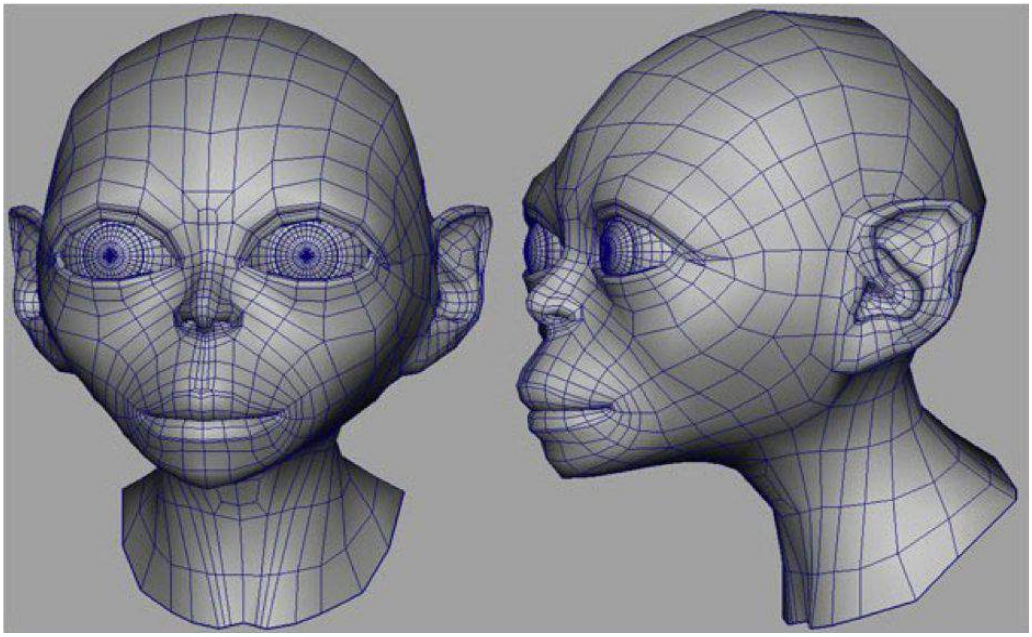
4.2 计算机视觉

所谓计算机视觉，即 compute vision，就是通过用计算机来模拟人的视觉工作原理，来完成模式分析，比如图像分类，分割，检测等。

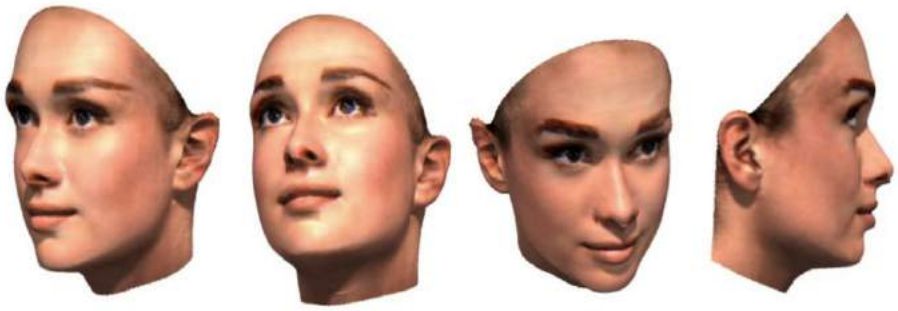
4.3 图形学

所谓计算机图形学(Computer Graphics，简称 CG)，是指使用数学算法将二维或三维图形转化为计算机显示器的栅格形式的科学。

简单地说，计算机图形学的主要研究内容就是研究如何在计算机中**表示图形**、以及利用计算机进行图形的计算，比如我们熟知的 **CG 制作**。



图形学中三维重建占了很大一部分比例，感兴趣可以了解更多。



3D Reconstruction

以上几个领域都是相互交叉，实际上没必要分的那么开，了解即可。

5 总结

基础的图像知识就这么多吧，虽然简单，但是一定要记牢了，免得以后还会回顾基础概念。

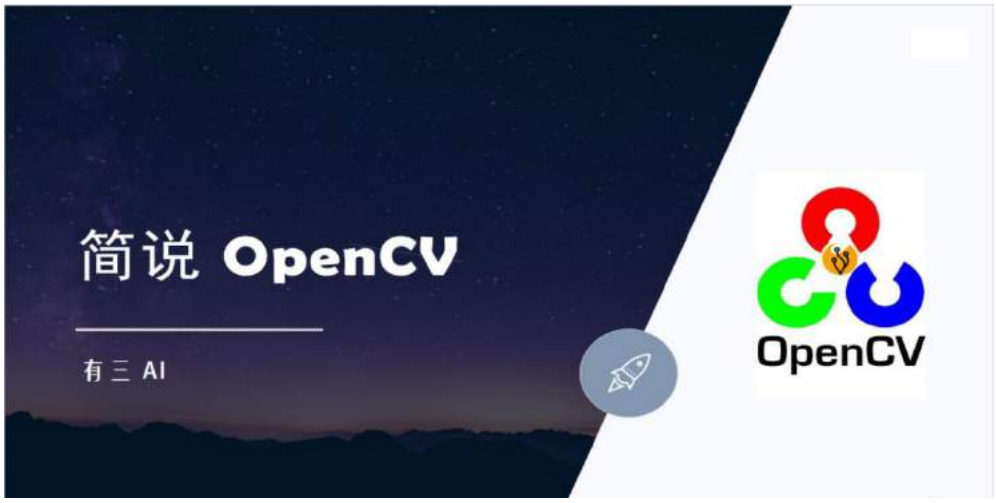
【AI 白身境】搞计算机视觉必备的 OpenCV 入门基础

本篇是**《AI 白身境》**的第五篇。曾经看过一个视频，树莓派自平衡机器人自动追着小球跑。不经让我脑子蹦出一个有趣的想法，可以做一个识别猫的机器人，让机器人跟着猫跑，有这样一个小东西陪伴喵星人一定很有意思。

不过，首先你要有一只猫，其次，这个机器人不仅要有一双会视觉处理的眼睛，还一定要有一个坚强的外壳，不然会被喵星人给拆了。

那机器人是如何完成处理图像和视频的各项任务呢？开源的计算机视觉包——OpenCV 会是你的最佳选择，本文给小白们做一个最简单的入门介绍。

作者|臧小满 言有三



很开心与大家分享一篇关于 OpenCV 的文章，重点阐述以下几个问题：

1.如何部署 OpenCV。

2.OpenCV 有哪些模块，可以做什么。

3.OpenCV 的基本数据结构的熟悉与使用。

希望看过文章后，你也可以开始玩转 OpenCV 之路。

1 什么是 OpenCV？

它是一款由 Intel 公司俄罗斯团队发起并参与和维护的一个计算机视觉处理开源软件库。

作为一款优秀的计算机视觉库，在诸多方面都有着卓越的表现：

1.1.编程语言

多数模块基于C++实现，少部分基于C语言实现，同时提供了Python、Ruby、MATLAB等语言的接口。

1.2.跨平台

可自由地运行在 Linux、Windows 和 Mac OS 等桌面平台，Android、 IOS、BlackBerray 等移动平台。

1.3.活跃的开发团队

目前已更新至 OpenCV4.0

1.4.丰富的 API

完善的传统计算机视觉算法，涵盖主流传统机器学习算法，同时添加了对深度学习的支持。

OpenCV 可以完成几乎所有的图像处理任务，下面是一个简要 list。

- 视频分析(Video analysis)
- 3D 重建(3D reconstruction)
- 特征提取(Feature extraction)
- 目标检测(Object detection)
- 机器学习(Machine learning)
- 计算摄影(Computational photography)
- 形状分析(Shape analysis)
- 光流算法(Optical flow algorithms)
- 人脸和目标识别(Face and object recognition)
- 表面匹配(Surface matching)
- 文本检测和识别(Text detection and recognition)

2 如何部署 OpenCV?

一般来说我们会使用 OpenCV 的 C++和 Python 版本，所以下面分别对其安装进行介绍，以 ubuntu 系统为例。

2.1 Ubuntu 安装 C++ OpenCV

安装 OpenCV 所需的库

```
sudo apt-get install build-essential
sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev3
sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev
```

下载最新 opencv 源码

```
unzip opencv-3.2.0.zip
cd ~/opencv-3.2.0
```

编译 OpenCV

```
cd ~/opencv-3.2.0
mkdir release
cd release
cmake -D CMAKE_BUILD_TYPE=RELEASE -D
CMAKE_INSTALL_PREFIX=/usr/local ..

make
sudo make install
```

一般来说，编译安装绝对不可能一次顺利完成，以下是几个常见的问题。

```
1，编译过程中 ippcv 下载失败，  解决问题的办法就是手动下载。
2，LAPACK 包 include 报错，  解决问题的办法就是在 cmake 之后马上修改对应 include 文件的路径   如果 make 失败后再修改则无效。
3，某些模块找不到，  通常是因为少了编译安装 contrib 模块。
```

2.2 Ubuntu 安装 Python-OpenCV

安装 opencv

```
pip3 install opencv-python
```

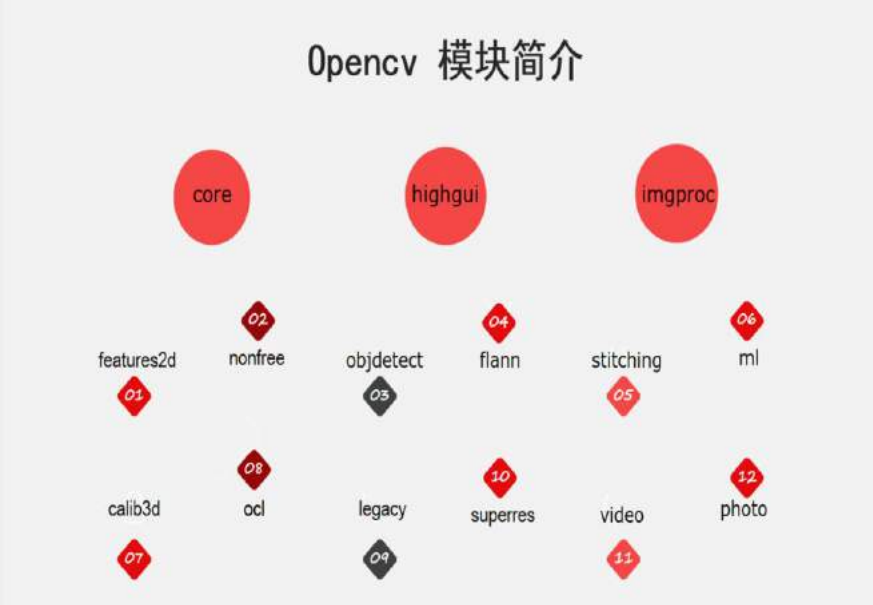
进入 python，导入 cv2

```
import cv2
```

3 OpenCV 模块简介

OpenCV 提供了许多内置的用于图像处理和计算机视觉相关操作的基础数据结构，它们都包含在 core 模块中，并且这些数据结构都已经针对速度和内存做了优化，下面以 4.0 版本为例进行介绍，参考 *https://docs.opencv.org/master/d9/df8/tutorial_root.html*。

Opencv 目录下”modules 目录”列出了 OpenCV 包含的各个模块，其中 **core、highgui、imgproc** 是最基础的模块。



core 模块实现了最核心的数据结构及其基本运算，如绘图函数、数组操作相关函数，与 OpenGL 的互操作等。

highgui 模块实现了视频与图像的读取、显示、存储等接口。

imgproc 模块实现了图像处理的基础方法，包括图像滤波、图像的几何变换、平滑、阈值分割、形态学处理、边缘检测、目标检测、运动分析和对象跟踪等。

对于图像处理其他更高层次的方向及应用，OpenCV 也有相关的模块实现

features2d 模块用于提取图像特征以及特征匹配，nonfree 模块实现了一些专利算法，如 sift 特征。

objdetect 模块实现了一些目标检测的功能，经典的基于 Haar、LBP 特征的人脸检测，基于 HOG 的行人、汽车等目标检测，分类器使用 Cascade Classification（级联分类）和 Latent SVM 等。

stitching 模块实现了图像拼接功能。

FLANN 模块（Fast Library for Approximate Nearest Neighbors），包含快速近似最近邻搜索 FLANN 和聚类 Clustering 算法。

ml 模块机器学习模块（SVM，决策树，Boosting 等等）。

photo 模块包含图像修复和图像去噪两部分。

video 模块针对视频处理，如背景分离，前景检测、对象跟踪等。

calib3d 模块即 Calibration（校准）3D，这个模块主要是相机校准和三维重建相关的内容。包含了基本的多视角几何算法，单个立体摄像头标定，物体姿态估计，立体相似性算法，3D 信息的重建等等。

G-API 模块包含超高效的图像处理 pipeline 引擎。

另外，原来在 opencv2 中的 shape, superres, videostab, viz 等模块被移动到 opencv_contrib 中，关于 opencv contrib，我们以后再详细介绍。

4 OpenCV 基本数据结构

OpenCv 提供了多种基本的数据类型，常用的 OpenCV 的基本数据结构有以下几种:

Mat 类

Point 类

Size 类

Rect 类

Scalar 类

Vec 类

Range 类

下面我们重点说一下 MAT 类。

4.1 Mat 类

要熟练使用 OpenCV，最重要的就是学会 Mat 数据结构，在 OpenCV 中 Mat 被定义为一个类，把它看作一个数据结构，以矩阵的形式来存储数据的。

Mat 有哪些常见的属性？

- dims：表示矩阵 M 的维度，如 2*3 的矩阵为 2 维，3*4*5 的矩阵为 3 维
- data：uchar 型的指针，指向内存中存放矩阵数据的一块内存
- rows，cols：矩阵的行数、列数
- type：表示了矩阵中元素的类型(depth)与矩阵的通道个数(channels)；命名规则为 CV_ +（位数）+（数据类型）+（通道数）

  其中：U（unsigned integer）-- 无符号整数

  S（signed integer）-- 有符号整数

  F（float）-- 浮点数

  例如 CV_8UC3，可拆分为：CV_：type 的前缀，

  8U：8 位无符号整数(depth)，C3：3 通道(channels)
- depth：即图像每一个像素的位数(bits)；这个值和 type 是相关的。例如 CV_8UC3 中 depth 则是 CV_8U。

- channels：通道数量，若图像为 RGB、HSV 等三通道图像，则 channels = 3；若图像为灰度图，则为单通道，则 channels = 1
- elemSize：矩阵中每一个元素的数据大小

  elemSize = channels * depth / 8

  例如：type 是 CV_8UC3，elemSize = 3 * 8 / 8 = 3bytes
- elemSize1：单通道的矩阵元素占用的数据大小

  elemSize1 = depth / 8

  例如：type 是 CV_8UC3，elemSize1 = 8 / 8 = 1bytes

4.2 其他数据类型

1.点 Point 类

包含两个整型数据成员 x 和 y，即坐标点

2.尺寸 Size 类

数据成员是 width 和 height，一般用来表示图像的大小，或者矩阵的大小

3.矩形 Rect 类

数据成员 x,y,width,height，分别代表这个矩形左上角的坐标点和矩形的宽度和高度

4.颜色 Scalar 类

Scalar_(_Tp v0, _Tp v1, _Tp v2=0, _Tp v3=0)

这个默认构造函数的四个参数分别表示 RGB+Alpha 颜色中的:

v0---表示 RGB 中的 B（蓝色）分量

v1---表示 RGB 中的 G（绿色）分量

v2---表示 RGB 中的 R（红色）分量

v3---表示 Alpha 是透明色分量

5.向量 Vec 类

一个“一维矩阵”

Vec<int,n>---就是用类型 int 和向量模板类做一个实例化。其中第一个参数 int 表示 Vec 中存储的为 int 类型；第二个参数 n 为一个整型值，表示 Vec 每个对象中存储 n 个 int 值，也就是 n 维向量(列向量)

6. Range 类

用于指定一个连续的子序列，例如一个轮廓的一部分，或者一个矩阵的列空间

5 基本 IO 操作

这里使用的是 python 接口

1. 图像读写

```
cv2.imread(文件名,显示控制参数)   #  读入图像
cv2.imshow(窗口名，图像名)      #显示图像
cv2.imwrite(文件地址,文件名)   #保存图像
cv2.namedWindow(窗口名)   #创建窗口
cv2.destroyAllWindows() #销毁窗口
cv2.waitKey(     [,delay])   #decay ＞ 0 等待 delay 毫秒
        #decay ＜ 0 等待键盘单击
        #decay =   0 无限等待
```

2.图像缩放

```
dst = cv2.resize(src,dsize,fx,fy)    #dsize 表示缩放大小
                                                                    #fx,
fy 缩放比例
```

3.图像翻转

```
dst = cv2.flip(src,flipCode)
                #flipCode=0 以 X 轴为对称轴的翻转
               #lipCode＞0 以 Y 轴为对称轴的翻转
               #flipCode＜0 对 X、Y 轴同时翻转
```

4.通道拆分与合并

```
b,g,r = cv2.split(图像)
b = cv2.split(图像)[通道数] #拆分
bgr = cv2.merge([b,g,r]) #合并
```

6 相关学习资料

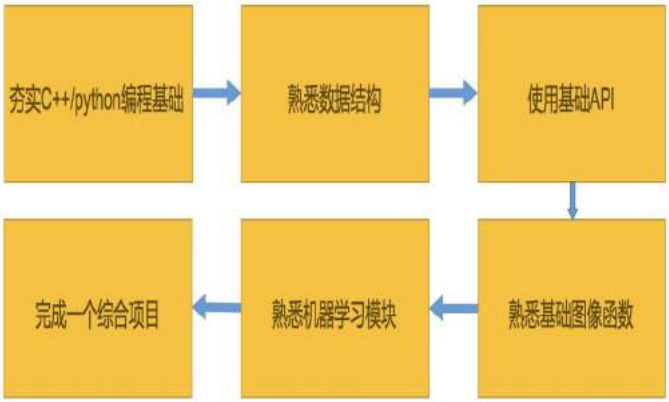
6.1 网络资料

OpenCV Docs 官方文档
https://docs.opencv.org/
OpenCV 官方 Github
https://github.com/opencv/opencv
OpenCV 中文教程
http://www.opencv.org.cn/opencvdoc/2.3.2/html/doc/tutorials/tutorials.html

6.2 中文书籍

Python 计算机视觉编程
OpenCV 3 计算机视觉：Python 语言实现
OpenCV 算法精解：基于 Python 与 C++

最后，推荐一下大家的 Opencv 学习路线。



7 总结

本文简单介绍了 OpenCV 框架，它是计算机视觉领域必须要熟练掌握的工具，这一期我们没有说具体的算法和模块，以后会开设**《OpenCV 专题》**讲述。

【AI 白身境】只会用 Python？g++，CMake 和 Makefile 了解一下

本篇是**《AI 白身境》**的第六篇，所谓白身，就是什么都不会，还没有进入角色。对于大部分小白来说，因为 python 用的太爽，以致于或许都没有听说过 CMake。python 是脚本语言，而当前大量的 AI 算法都部署在移动端嵌入式平台，需要使用 c/java 语言，因此熟悉 CMake 和 Makefile 也是必备的基础。

作者 | 汤兴旺 言有三
编辑 | 汤兴旺 言有三

1 g++必备基础

在学习 CMake 和和 Makefile 之前我们先学下 g++这个工具，大家或许会问为什么要学 g++，不应该直接学 CMake 和 Makefile 吗。实际上如果你不掌握 g++根本就不会写 Makefile，因为它实际上就是对 g++代码的整理，有了 Makefile，执行程序会更加快速方便。另外 CMake 就是为了简化 Makefile 的编写，它可以自动生成 Makefile。

1.1 安装 g++

我们在安装 g++之前可以看一下自己是否已经安装了 g++，因为 ubuntu 安装后就默认安装了 g++，下面命令可查看自己 g++版本。

Tips：如果不想作死，就不要手贱去降级或者升级 g++版本。

```
g++ --version
```

```
tangxingwang@tangxingwang-Lenovo-Y430P:~$ g++ --version
g++ (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

tangxingwang@tangxingwang-Lenovo-Y430P:~$
```

因为我已经安装了 g++，出现了上面安装的版本号。如果你出现了上面信息，就不需要再安装了，没有的话，用下面的命令即可完成安装。

```
sudo apt-get install g++
```

安装好后也可以通过 g++ --version 查看是否安装成功

1.2 编译流程

现在我们已经安装好了 g++，接下来通过写一个简单的程序来看看整个的编译流程。

我们通过 vim 创建一个 test.cpp 文件，测试的代码如下：

```
#include <iostream>
using namespace std;
int main()
{
        cout << "Hello, world!" <<endl;
        return 0;
}
```

测试代码完成后，我们来进行下编译，打开终端，在终端输入 g++ 文件名即可，在这个程序中就是下面命令：

```
g++  test.cpp
```

注意这里的文件名是包括路径的，要是不知道文件路径的话可以在敲完 g++和空格之后直接把文件拖进去，系统会自动添加文件路径。

在终端完成上面的命令后，你发现并没有任何输出，但这时候你去主文件夹下（默认主文件夹）看下会发现有个 a.out 文件

现在你再在终端输入下面命令就能看到结果。

```
./a.out
```

```
tangxingwang@tangxingwang-Lenovo-Y430P:~$ vim test.cpp
tangxingwang@tangxingwang-Lenovo-Y430P:~$ g++ test.cpp
tangxingwang@tangxingwang-Lenovo-Y430P:~$ ./a.out
Hello, world!
tangxingwang@tangxingwang-Lenovo-Y430P:~$
```

接下来我来解释下这个.out 文件，实际上这是个经过相应的链接产生的可执行文件。还有个.o 文件，它是个中间文件，一般是通过编译的但还未链接。我们通过看看 g++在执行编译工作的时候的流程，你就会有更好的理解。如下：

1.预处理，生成.i 的文件
2.将预处理后的文件转换成汇编语言，生成.s 文件
3.将汇编变为目标代码(机器代码)，生成.o 的文件
4.连接目标代码,生成可执行程序

对于这个流程，我们结合上面的例子，再详细介绍下，如下：

1.预处理阶段

首先在终端输入下面代码：

```
g++ -E test.cpp > test.i
```

预处理后的文件在 linux 下以.i 为后缀名，这个过程是用来激活预处理，执行完命令后，你会发现主文件夹下多了一个 test.i 文件

这一步（预处理）主要做了宏的替换，和注释的消除。

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)

  extern wistream wcin;
  extern wostream wcout;
  extern wostream wcerr;
  extern wostream wclog;



  static ios_base::Init __ioinit;


}
# 2 "test.cpp" 2

# 2 "test.cpp"
using namespace std;
int main()
{
    cout << "Hello, world!" <<endl ;
    return 0;
}
tangxingwang@tangxingwang-Lenovo-Y430P:~$
```

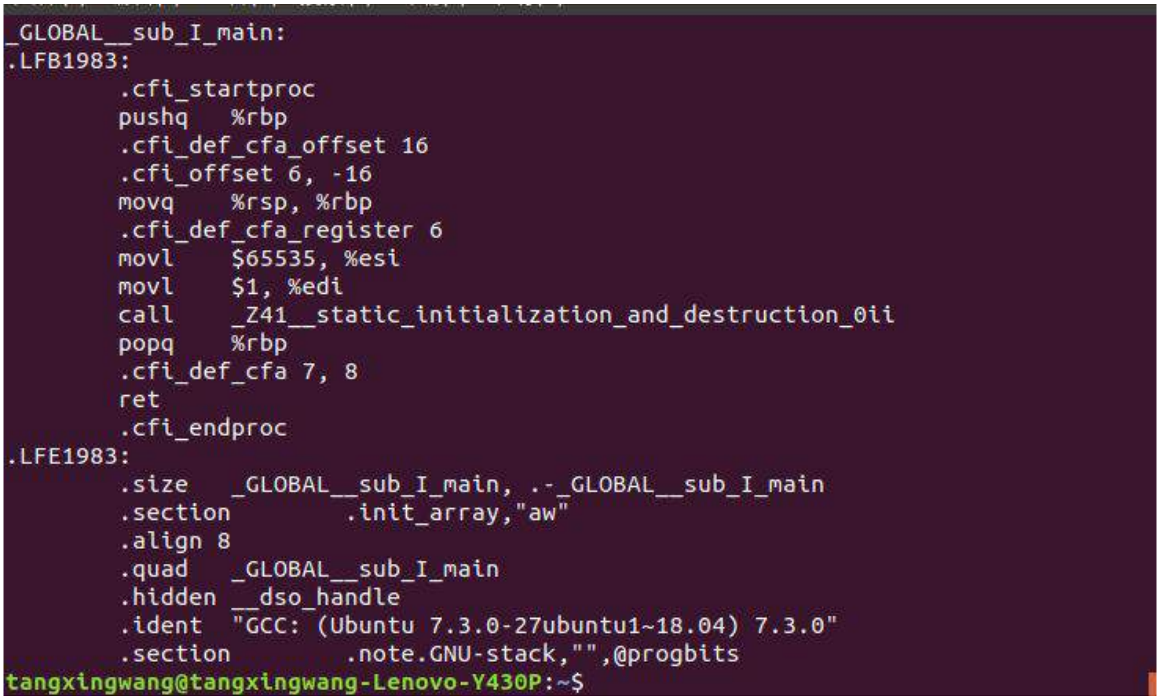
上图是 test.i 文件的最后部分，可以看见宏的替换和注释的消除。

2.将预处理后的文件转换成汇编语言

在终端输入下面代码：

```
g++ -S test.cpp
```

这一步主要就是生成 test.s 文件，.s 文件表示汇编文件，用编辑器打开就都是汇编指令。下图是 test.s 文件的一部分。

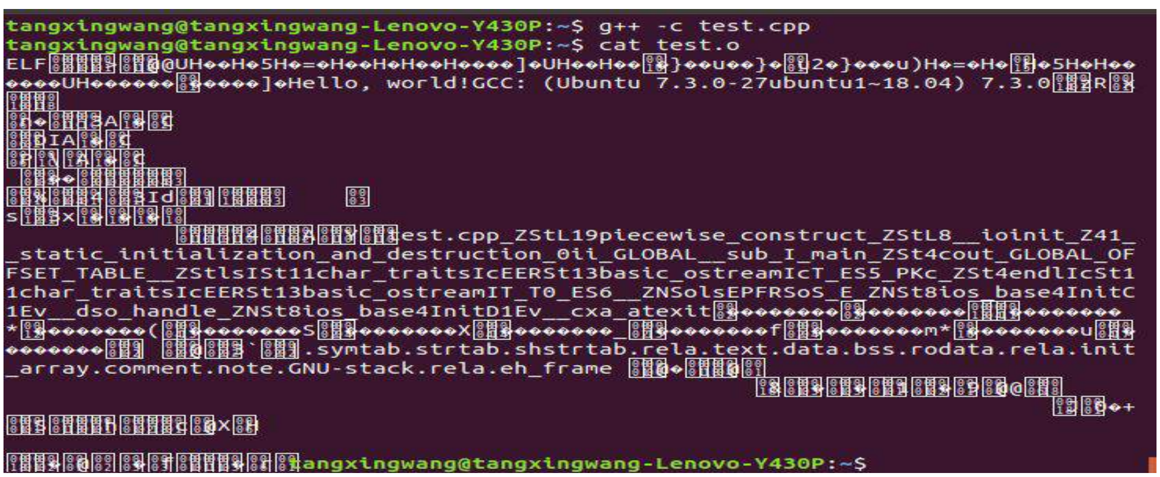


3.将汇编语言变为目标代码(机器代码)

在终端输入下面代码：

```
g++ -c test.cpp
```

这一步就是生成目标文件，用编辑器打开就都是二进制机器码。

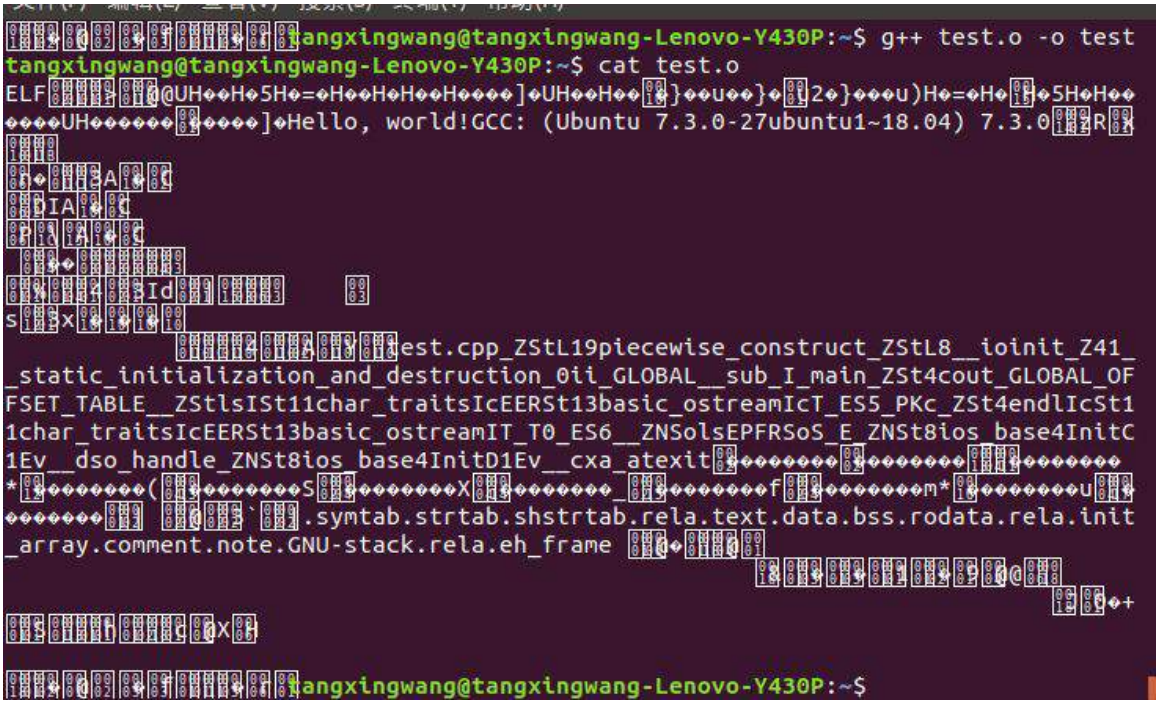


4.链接目标代码，生成可执行程序

在终端输入下面代码：

```
g++ test.o -o test
```

在这一步中生成的可执行程序名为 test，如果执行命令 g++ test.o 这样默认生成 a.out



最后我们再看下这个过程中产生的所有文件，如下：

这就是编译的整个过程，你掌握了吗，这个过程对于后面编写 Makefile 非常重要，一定要深刻理解。

2 Makefile 必备基础

上面我们对 g++和编译过程进行了介绍，现在我们继续学习如何编写 Makefile。

2.1 Makefile 介绍

Makefile 描述了整个工程的编译、链接等规则，它定义了一系列规则来指定哪些文件需要编译以及如何编译、需要创建哪些库文件以及如何创建这些库文件、如何产生我们想要的可执行文件。

而且 Makefile 可以有效的减少大工程中需要编译和链接的文件，只编译和链接那些需要修改的文件，可以说使用 Makefile，整个工程都可以完全自动化编译。

2.2 Makefile 基本格式

```
target ... : prerequisites ...
    command
    ...
    ...
```

target - 目标文件，可以是 Object File，也可以是可执行文件

prerequisites - 生成 target 所需要的文件或者目标

command - make 需要执行的命令(任意的 shell 命令)，Makefile 中的命令必须以 [tab] 开头

2.3 Makefile 语法

Makefile 包含了五个重要的东西：显示规则、隐晦规则、变量定义、文件指示和注释。详细解释如下：

1. 显示规则：
通常在写 makefile 时使用的都是显式规则，这需要指明 target 和 prerequisite 文件。一条规则可以包含多个 target，这意味着其中每个 target 的 prerequisite 都是相同的。当其中的一个 target 被修改后，整个规则中的其他 target 文件都会被重新编译或执行。

2. 隐晦规则：
make 的自动推导功能所执行的规则

3. 变量的定义：
Makefile 中定义的变量，一般是字符串

4. 文件指示：
Makefile 中引用其他 Makefile；指定 Makefile 中有效部分；定义一个多行命令

5. 注释：
Makefile 只有行注释 "#"，如果要使用或者输出"#"字符，需要进行转义，"\#

2.4 Makefile 简单实例

尽管上面介绍了许多 Makefile 的知识点，但我相信一定你很晕，接下来我通过一个实例来说明如何编写 Makefile。

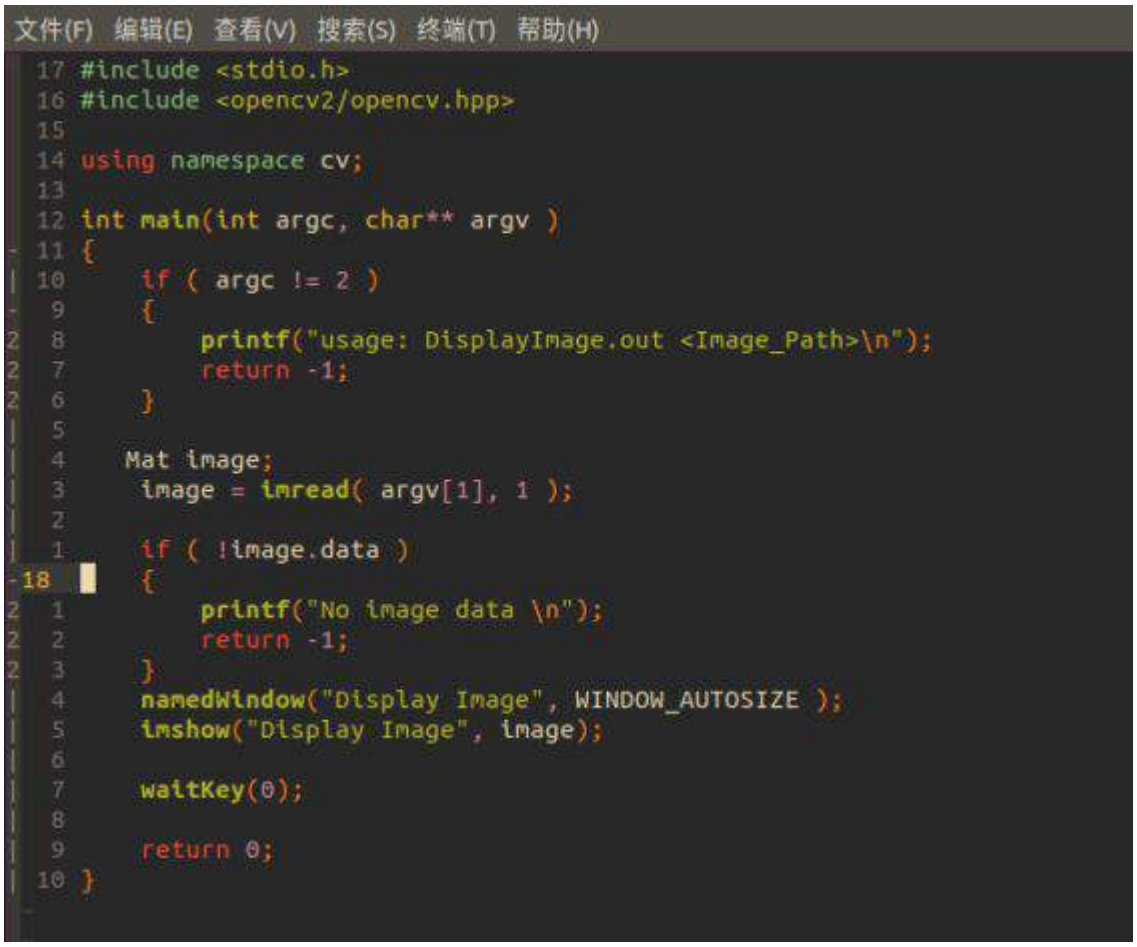
2.4.1 准备程序文件

我们使用 opencv 对下面这只可爱的猫进行读取显示。

在这里我们用 c++和 opencv 对图片进行读取和显示，程序保存在 DisplayImage.cpp 这个文件里，代码如下：

```
#include <stdio.h>
#include <opencv2/opencv.hpp>
using namespace cv;
int main(int argc, char** argv )
{
    if ( argc != 2 )
    {
        printf("usage: DisplayImage.out <Image_Path>\n");
        return -1;
    }
```

```
    Mat image;
     image = imread( argv[1], 1 );
     if ( !image.data )
     {
          printf("No image data \n");
          return -1;
     }
     namedWindow("Display Image", WINDOW_AUTOSIZE );
     imshow("Display Image", image);
     waitKey(0);
     return 0;
}
```



上面我们已经准备好了.cpp 文件，现在我们来编写 Makefile 进而进行编译，程序如下：

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
13 export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/home/tangxingwang/cv_sdk/lib/pkgco
   nfig
12
11 CXXFLAGS:= $(shell pkg-config --cflags --libs opencv)
10
 9 DisplayImage:DisplayImage.o
 8         g++ DisplayImage.o -o DisplayImage          $(CXXFLAGS)
 7 DisplayImage.o:DisplayImage.cpp
 6         g++ -c DisplayImage.cpp -o DisplayImage.o       $(CXXFLAGS)
 5 clean:
 4         rm *o test
 3
```

现在我来解释下应该如何编写这个 Makefile，对于编写 Makefile 我建议从下往上写。步骤如下：

1.编写 clean

这一步在 Makefile 中基本差不多，它的作用就是删除所有的.o 文件和可执行文件。为什么这样做呢?我举个例子说明下，如果你有 100 个.cpp 文件，经过编译后会得到一个可执行文件。在这个过程中我们会得到许多不必要的文件，例如 100 个.o 文件，但这个文件又没有用，如果用 rm 的话那就太麻烦了，所以我们用了 clean，它可以很轻松完成这个任务。另外请注意 Makefile 文件在执行时不会执行 clean 这个命令，需要我们调用才会执行，即 make clean。clean 代码如下：

```
 5 clean:
 4         rm *o test
```

2.编写目标文件 1：依赖文件 1

目标文件就是你想得到的文件，依赖文件就是你目前所拥有的东西。在本实例中我们现在拥有 DisplayImage.cpp，所以 DisplayImage.cpp 是依赖文件，我们想得到 DisplayImage.o，所以它是目标文件。代码如下：

```
 7 DisplayImage.o:DisplayImage.cpp
 6         g++ -c DisplayImage.cpp -o DisplayImage.o       $(CXXFLAGS)
```

3.编写目标文件 2：依赖文件 2

这一步的依赖文件 2 实际就是第二步的目标文件 1，在第二步我们通过 DisplayImage.cpp 得到了 DisplayImage.o，现在我们需要通过 DisplayImage.o 得到可执行文件 DisplayImage。所以在这一步目标文件是 DisplayImage，依赖文件是 DisplayImage.o，代码如下：

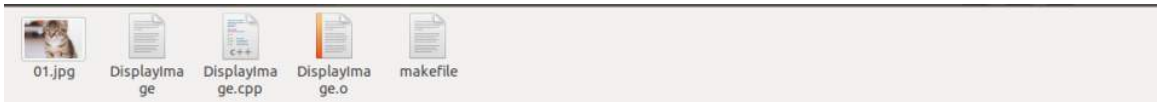
```
 9 DisplayImage:DisplayImage.o
 8         g++ DisplayImage.o -o DisplayImage          $(CXXFLAGS)
```

4.应用 opencv 库和头文件

```
13 export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/home/tangxingwang/cv_sdk/lib/pkgco
   nfig
12
11 CXXFLAGS:= $(shell pkg-config --cflags --libs opencv)
```

这一步就需要根据自己计算机来配置了，对于我们初学者来说挺麻烦的，可以自己尝试下。有问题可以联系我们。

编写完 makefile 后，我们在终端 make 下就行了。下面编译后的文件：



最后在终端输入下面代码即可显示图片。

```
./DisplayImage 01.jpg
```

总体来说编写 Makefile 可以按照这个套路写，多写几次就会了。

3 CMake 必备基础

说完 Makefile，我们再说下 CMake。CMake 是一个跨平台的编译(Build)工具，可以用简单的语句来描述所有平台的编译过程，其是在 make 基础上发展而来的，早期的 make 需要程序员写 Makefile 文件，进行编译，而现在 CMake 能够通过对 cmakelists.txt 的编辑，轻松实现对复杂工程的组织。下面我带大家学习下 CMake 的基础知识。

3.1 安装 CMake

首先我们看看如何在自己的 linux 系统(我的系统 Ubuntu18.04)下安装 CMake。方法如下：

```
sudo apt-get install cmake
```

输入上面命令后实际上就安装成功了，可以通过下面命令来检查：

```
cmake --version
```

如果你的界面如下图所示即说明安装成功。

```
tangxingwang@tangxingwang-Lenovo-Y430P:~$ cmake --version
cmake version 3.10.2
```

3.2 CMake 编译流程

成功安装好 CMake 后我们再来说说如何在 linux 平台下使用 CMake 生成 Makefile 并编译的流程，如下：

1.编写 CMake 配置文件 CMakeLists.txt，我们可以认为 CMakeLists.txt 就是 CMake 所处理的"代码"。

2.执行命令 cmake path 生成 Makefile,其中 path 是 CMakeLists.txt 所在的目录。
3.使用 make 命令进行编译。

3.3 使用 CMake 编译程序

我们通过一个关于 opencv 读取图片的程序，让大家更好的理解整个 CMake 的编译过程。

3.3.1 准备程序文件

这里程序准备可以按照第二部分 makefile 那里准备。最后文件目录结构如下：

```
├── build
├── CMakeLists.txt
├──  DisplayImage.cpp
```

opencv 读取图片的程序写完后，我们需要编写 CMake 处理的代码了，即 CMakeLists.txt。

3.3.2 编写 CMakeLists.txt

现在我们编写 CMakeLists.txt 文件，该文件实际上放在哪里都可以，只要编写的路径能够正确指向就好了，CMakeLists.txt 文件内容如下所示：

```
cmake_minimum_required(VERSION 2.8)
project( DisplayImage )
find_package( OpenCV REQUIRED )
add_executable( DisplayImage DisplayImage.cpp )
target_link_libraries(          DisplayImage          ${OpenCV_LIBS}          )
```

```
cmake_minimum_required(VERSION 2.8)
project( DisplayImage )
find_package( OpenCV REQUIRED )
add_executable( DisplayImage DisplayImage.cpp )
target_link_libraries( DisplayImage ${OpenCV_LIBS} )
```

看到这些代码是不是很闷逼，为了让大家明白 CMakeLists.txt 文件内容，接下来我说一下 Cmake 的一些常用命令，你就能很好的理解上面的代码了。

1）cmake_minimum_required 命令
命令语法：cmake_minimum_required(VERSION major[.minor[.patch[.tweak]]][FATAL_ERROR])
命令简述：用于指定需要的 CMake 的最低版本
使用范例：cmake_minimum_required(VERSION 2.8)

2）project 命令
命令语法：project(<projectname> [languageName1 languageName2 …])
命令简述：用于指定项目的名称，一般和项目的文件夹名称对应
使用范例：project(DisplayImage)

3）aux_source_directory 命令
命令语法：aux_source_directory(<dir> <variable>)
命令简述：用于将 dir 目录下的所有源文件的名字保存在变量 variable 中
使用范例：aux_source_directory(src DIR_SRCS)

4）add_executable 命令
命令语法：add_executable(<name> [WIN32] [MACOSX_BUNDLE][EXCLUDE_FROM_ALL] source1 source2 … sourceN)
命令简述：用于指定从一组源文件 source1 source2 … sourceN 编译出一个可执行文件且命名为 name
使用范例:add_executable(DisplayImage DisplayImage.cpp)

5）target_link_libraries 命令
命令语法：target_link_libraries(<target> [item1 [item2 […]][[debug|optimized|general]] …)
命令简述：用于指定 target 需要的链接 item1 item2 …。这里 target 必须已经被创建，链接的 item 可以是已经存在的 target（依赖关系会自动添加）
使用范例：target_link_libraries(DisplayImage ${OpenCV_LIBS})

6）add_subdirectory 命令
命令语法：add_subdirectory(source_dir [binary_dir] [EXCLUDE_FROM_ALL])
命令简述：用于添加一个需要进行构建的子目录
使用范例：add_subdirectory(Lib)

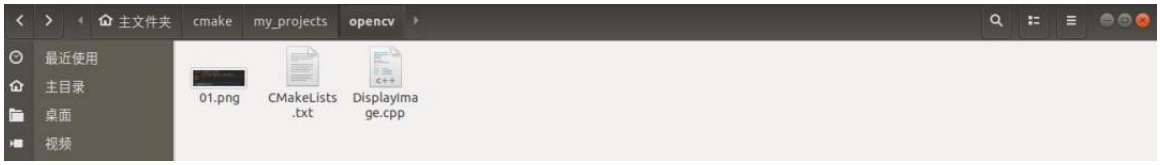
7）include_directories 命令
命令语法：include_directories([AFTER|BEFORE] [SYSTEM] dir1 dir2 …)
命令简述：用于设定目录，这些设定的目录将被编译器用来查找 include 文件
使用范例：include_directories(${PROJECT_SOURCE_DIR}/lib)

像这样的命令还有很多，如 find_package（）寻找使用第三方库等，这些都需要我们平时多加积累。给大家一个查询命令的方法，大家可以多去看 cmake 官网的 help，链接如下：

https://cmake.org/cmake/help/v2.8.8/cmake.html#section_Commands

3.3 编译和运行程序

现在 CMakeLists.txt 文件已经编写好了，意味着我们的工作即将进入尾声。现在看看我们的文件结构目录，如下图：



接下来我们就需要进行编译了。编译的过程相对于 CMakeLists.txt 文件的编写是很简单的，只有两步，如下

```
cmake
make
```

其中 cmake 命令将 CMakeLists.txt 文件转化为 make 所需要的 makefile 文件，最后用 make 命令编译源码生成可执行程序或共享库。对于我们这个实例，编译如下：

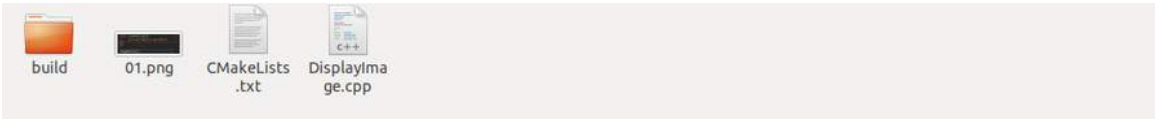
首先我们在命令行输入 **cmake .**(注意 cmake 和.之间有空格)，表明 Cmakelist.txt 文件在当前目录下。

```
tangxingwang@tangxingwang-Lenovo-Y430P:~/cmake/my_projects/opencv/build$ cmake .
.
-- The C compiler identification is GNU 7.3.0
-- The CXX compiler identification is GNU 7.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenCV: /usr/local/opencv4 (found version "4.0.1")
-- Configuring done
-- Generating done
-- Build files have been written to: /home/tangxingwang/cmake/my_projects/opencv
/build
```

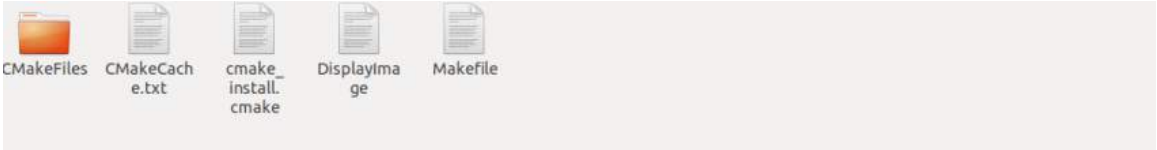
接下来在命令行输入 **make**

```
tangxingwang@tangxingwang-Lenovo-Y430P:~/cmake/my_projects/opencv/build$ make
Scanning dependencies of target DisplayImage
[ 50%] Building CXX object CMakeFiles/DisplayImage.dir/DisplayImage.cpp.o
[100%] Linking CXX executable DisplayImage
[100%] Built target DisplayImage
```

这样我们就编译成功了，我们看下编译后的文件目录



解释下这个 build 文件夹，由于 cmake 后会生成很多编译的中间文件以及 makefile 文件，所以一般建议新建一个新的目录，专门用来编译，这就是这里的 build，打开 build 后，里面的文件如下：



到这里，我们不禁要问怎么没有图片显示呢，别急，在 build 目录下的命令行输入下面命令即可显示图片，这就是生产的 DisplayImage 可执行文件。

```
./DisplayImage ../01.jpg
```

```
tangxingwang@tangxingwang-Lenovo-Y430P:~/cmake/my_projects/opencv/build$ ./Displ
ayImage ../01.jpg
```

到这里，关于 CMake 的一些基本操作就介绍的差不多了，其实对于 CMake 的学习我认为必须在实例中多加应用，才能更好的掌握，因为它的复杂命令太多了。

4 总结

CMake 和 Makefile 的基础我们就介绍完了，对于这两个工具其实不是一时就能学会的，需要大量的实践积累才能游刃有余。

【AI 白身境】学深度学习你不得不知的爬虫基础

本篇是**《AI 白身境》**的第七篇，所谓白身，就是什么都不会，还没有进入角色。

对于深度学习，一个好的数据集可以说非常重要的，但是通常情况下我们并没有大量的数据，因此我们有必要掌握一定的爬虫知识，从而更好的准备训练数据集。

作者|汤兴旺 言有三

1 前端网页基础

在介绍爬虫之前我们先说下网页基础，理解前端网页的一些基础知识对于学习爬虫是很有必要的，它是爬虫的基础之一。

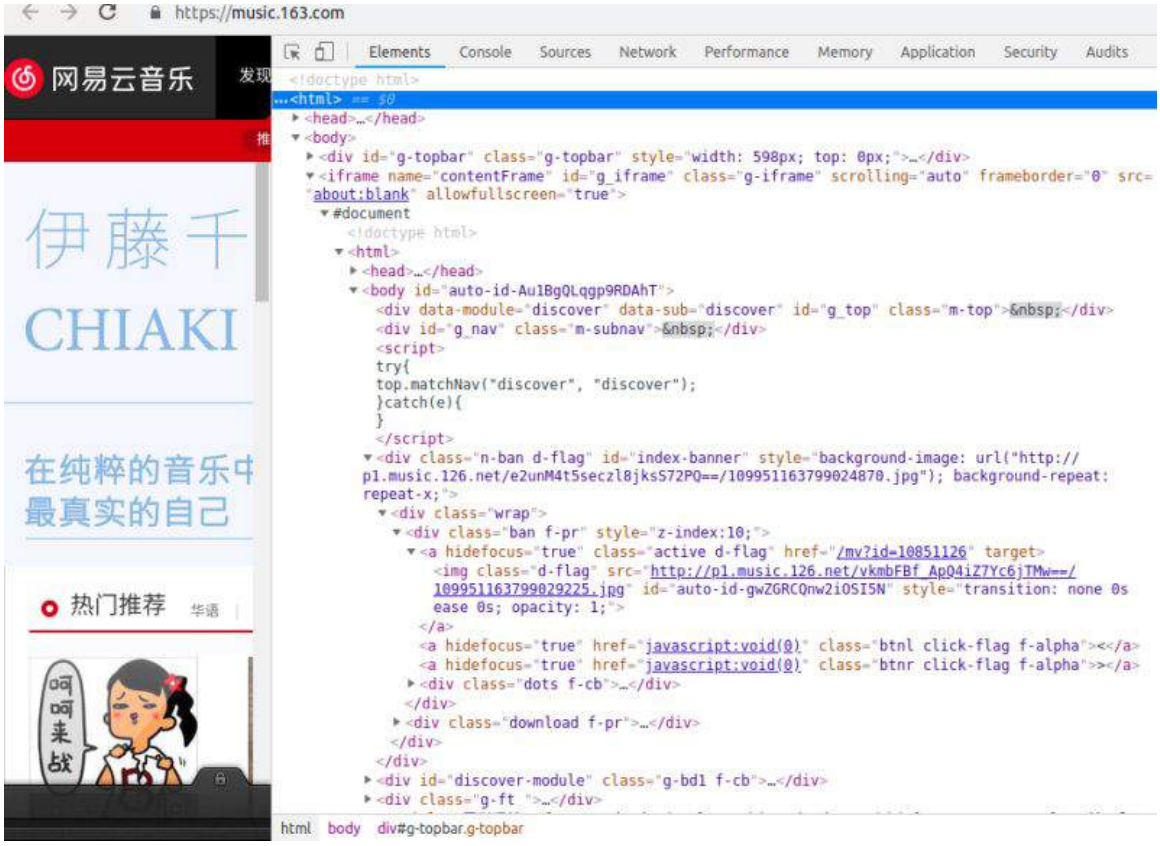
1.1 网页构成

通常情况下我们看到的网页由三部分组成，分别是 HTML、CSS 和 JavaScript，接下来我分别介绍它们。

1.1.1 HTML

HTML，全称 Hyper Text Markup Language，也就是“超文本链接标示语言”。但它不是一种编程语言，而是一种标记语言。我们通常看到的网页就是 HTML 使用标记标签来描述的。在 HTML 中，通常不同类型的文字通过不同类型的标签来表示。如图片用 img 标签表示，视频用 video 标签表示，段落用 p 标签表示。

现在我们看下网易云音乐的源代码，如下图所示：

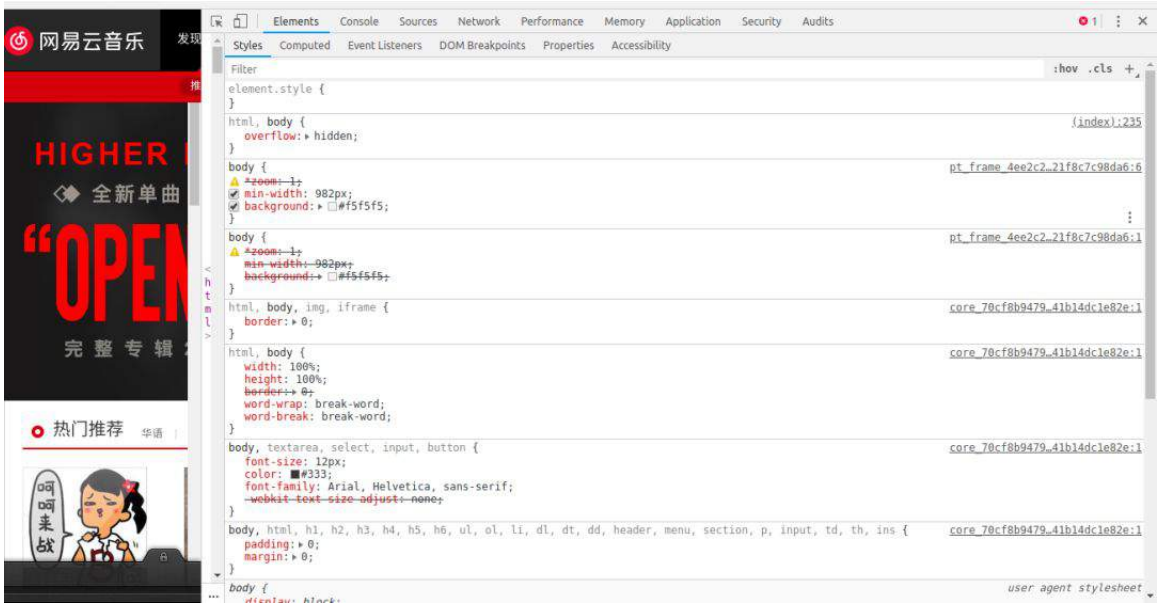


获取这个网页的源代码很简单，打开网页后，按下F12键就会出现这个源代码了。通过观察这个 HTML 我们会发现整个网页就是由各种标签嵌套组合而成的，从而形成了整个网页的架构。

1.1.2 CSS

从上面的介绍我们知道 HTML 定义了网页的架构，可以认为是一个框架，但若只有HTML，那么这样的网页就太简陋了，为了让我们的网页更加好看点，我们就需要用 CSS。

CSS，全称 Cascading Style Sheets，即层叠样式表。“层叠”是指当在 HTML 中引用了数个样式文件，并且样式发生冲突时，浏览器能依据层叠顺序处理。“样式”指网页中文字大小、颜色、元素间距、排列等格式。我们来看看网易云音乐的 CSS，如下所示。



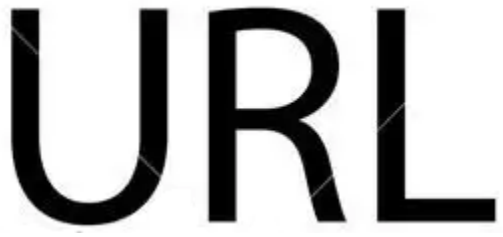
1.1.3 JavaScript

上面介绍的 HTML 和 CSS 只能展现一种静态信息，缺乏交互性。但我们在网页里通常会看到一些交互和动画效果，如提示框、轮播图等，这些动态信息通常就是通过 JavaScript 完成的。它的出现使得用户与信息之间不只是一种浏览与显示的关系，而是实现了一种实时、动态、交互的页面功能。

这就是网页构成的一些基本知识，你掌握了吗？

1.2 URL

爬虫最主要的处理对象是什么？那毫无疑问肯定是 URL，爬虫的实质就是根据 URL 地址取得所需要的文件内容，然后对它进行进一步的处理。所以说准确理解 URL 也是理解网络爬虫的基础之一。



URL，全称是 Uniform Resource Locator,通俗地说，URL 是 Internet 上描述信息资源的字符串，主要用在各种 WWW 客户程序和服务器程序上。URL 也有它特定的格式，其格式由三部分组成，如下：

1.第一部分是协议(或称为服务方式)。

2.第二部分是存有该资源的主机 IP 地址(有时也包括端口号)。
3.第三部分是主机资源的具体地址，如目录和文件名等。

通常第一部分和第二部分用“://”符号隔开，第二部分和第三部分用“/”符号隔开。另外第一部分和第二部分是不可缺少的，第三部分有时可以省略。

我们通过一个 URL 的一个小例子来解释下上面的三部分，下面是 NBA 中国官方网站湖人队网页的 URL：

```
http://china.nba.com/lakers/
```

http 这个是协议，也就是 HTTP 超文本传输协议，它是 URL 的第一部分；china.nba.com 这个是网站名，由服务器名和域名构成，它是 URL 的第二部分；lakers 就是存放网页的根目录，是 URL 的第三部分。

这就是 URL 的一些基础知识，希望大家深刻理解。

通过上面的介绍，相信你对网页的基础知识也有了大致的了解，下面我们开始学习爬虫相关库的一些基础知识。

2 python 爬虫库

了解了网页的一些基础知识后，我们继续来学习下 python 爬虫的一些库，通过前面第三篇文章《AI 白身境学习 AI 必备的 python 基础》我们都知道 python 有许多库，如 NumPy，matplotlib 等，针对爬虫它有个自带的库 urllib。

2.1 urllib 介绍

urllib 是 python 自带的一个主要用来爬虫的标准库，无需安装可以直接用，它能完成如下任务：网页请求、响应获取、代理和 cookie 设置、异常处理和 URL 解析，可以说要想学会爬虫，必须要学会它。

2.2 urllib 基础用法

我们已经知道 urllib 能完成网页请求、响应获取等许多任务，现在我来介绍下它的基本用法。

2.2.1 发起 GET/POST 请求

在用 urllib 实现 GET 和 POST 之前，我们先解释下什么是 GET？什么是 POST？它们的区别又是啥？

GET 和 POST 实际上就是 HTTP 请求的两种基本方法，通常 GET 是从指定的资源请求数据，而 POST 是向指定的资源提交要被处理的数据。我们再看看它的区别是啥，请看下面表格：

	GET	POST
后退按钮/刷新	无害	数据会被重新提交（浏览器应该告知用户数据会被重新提交）。
书签	可收藏为书签	不可收藏为书签
缓存	能被缓存	不能缓存
编码类型	application/x-www-form-urlencoded	application/x-www-form-urlencoded 或 multipart/form-data。为二进制数据使用多重编码。
历史	参数保留在浏览器历史中。	参数不会保存在浏览器历史中。
对数据长度的限制	是的。当发送数据时，GET 方法向 URL 添加数据；URL 的长度是受限制的（URL 的最大长度是 2048 个字符）。	无限制。
对数据类型的限制	只允许 ASCII 字符。	没有限制。也允许二进制数据。
安全性	与 POST 相比，GET 的安全性较差，因为所发送的数据是 URL 的一部分。 在发送密码或其他敏感信息时绝不要使用 GET！	POST 比 GET 更安全，因为参数不会被保存在浏览器历史或 web 服务器日志中。
可见性	数据在 URL 中对所有人都是可见的。	数据不会显示在 URL 中。

哈哈，你现在看到这些肯定很闷逼。

我们从头（HTTP)来分析下，我们已经知道 HTTP 是基于 TCP/IP 的关于数据如何在万维网中如何通信的协议。HTTP 的底层是 TCP/IP，所以 GET 和 POST 的底层也是 TCP/IP，也就是说，GET/POST 都是 TCP 连接，GET 和 POST 能做的事情是一样的。

那它们的区别体现在哪呢？对于 GET 方式的请求，浏览器会把 http header 和 data 一并发送出去；而对于 POST，浏览器先发送 header，服务器响应后，浏览器再发送 data。

也就是说，在大万维网世界中，TCP 就像汽车，我们用 TCP 来运输数据，HTTP 给汽车运输设定了好几个运输方式，有 GET，POST 等。GET 只需要汽车跑一趟就把货送到了，而 POST 得跑两趟，第一趟，先去和服务器打个招呼“嗨，我等下要送一批货来，你们打开门迎接我”，然后再回头把货送过去。

你现在明白它们的区别了吗？我们再看看 urllib 是如何使用这两个方法的。

在 urllib 中有个 request 这个模块，它主要是来负责构造和发起网络请求。它有个 urlopen()访问方法，默认的访问方法是 GET，我们在 urlopen()方法中传入字符串格式的 url 地址后，此方法会访问目标网址，然后返回访问的结果。请看下面的实例：

```
from urllib import request
url = "https://zhuanlan.zhihu.com/p/20751612"
html = request.urlopen(url).read().decode("utf-8")
print(html)
```

```
3 from urllib import request
2 url = "https://zhuanlan.zhihu.com/p/20751612"
1 html = request.urlopen(url).read().decode("utf-8")
5 print(html)
```

```
<!doctype html>
<html lang="zh" data-hairline="true" data-theme="light"><head><meta charSet="utf
-8"/><title data-react-helmet="true">当我在现场告别科比，我告别了什么？ - 知乎</
title><meta name="viewport" content="width=device-width,initial-scale=1,maximum-
scale=1"/><meta name="renderer" content="webkit"/><meta name="force-rendering" c
ontent="webkit"/><meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"/>
<meta name="google-site-verification" content="FTeR0c8arOPKh8c5DYh_9uu98_zJbaWw5
3J-Sch9MTg"/><title>知乎 - 有问题上知乎</title><meta data-react-helmet="true" pr
operty="description" content="这是我第一次在现场看NBA，然而它是科比的最后一场比
赛。 很难装作科比占据了我的青春，虽然我在现场。 远赴洛杉矶看一场科比的退役赛，听
上去好像只有狂热粉丝才干得出- - 更何况，那张去年8月份入手时不到3000元… "/><meta
data-react-helmet="true" property="og:title" content="当我在现场告别科比，我告别
了什么？ "/><meta data-react-helmet="true" property="og:url" content="https://zhu
anlan.zhihu.com/p/20751612"/><meta data-react-helmet="true" property="og:descrip
tion" content="这是我第一次在现场看NBA，然而它是科比的最后一场比赛。 很难装作科
比占据了我的青春，虽然我在现场。 远赴洛杉矶看一场科比的退役赛，听上去好像只有狂
热粉丝才干得出- - 更何况，那张去年8月份入手时不到3000元… "/><meta data-react-hel
met="true" property="og:image" content="https://pic3.zhimg.com/cf170b556aa59de65
62268a6b208fb40_b.jpg"/><meta data-react-helmet="true" property="og:type" conten
t="article"/><meta data-react-helmet="true" property="og:site_name" content="知
乎专栏"/><link data-react-helmet="true" rel="apple-touch-icon" href="
-- More --
```

urlopen()方法请求返回的对象类型为 HTTPResponse 对象。

```
from urllib import request
url = "https://zhuanlan.zhihu.com/p/20751612"
html = request.urlopen(url)
print(type(html))
```

```
#!env python3
from urllib import request
url = "https://zhuanlan.zhihu.com/p/20751612"
html = request.urlopen(url)
print(type(html))
```

```
NORMAL | 1.py[+] |              python | utf-8[unix] | 100%[5,17] | 01/17-21:05
<class 'http.client.HTTPResponse'>
```

返回的状态为 200，即返回数据。

`print(html.status)`

```
#!env python3
from urllib import request
url = "https://zhuanlan.zhihu.com/p/20751612"
html = request.urlopen(url)
print(html.status)
```

```
NORMAL | 1.py[+] |              python | utf-8[unix] | 100%[5,17] | 01/17-21:11
200
```

返回的数据会是 bytes 的二进制格式，所以需要 decode()一下，转换成字符串格式。

`print(html.read().decode("utf-8")`

```
                                                                    )
<!doctype html>
<html lang="zh" data-hairline="true" data-theme="light"><head><meta charSet="utf
-8"/><title data-react-helmet="true">当我在现场告别科比，我告别了什么？ - 知乎</
title><meta name="viewport" content="width=device-width,initial-scale=1,maximum-
scale=1"/><meta name="renderer" content="webkit"/><meta name="force-rendering" c
ontent="webkit"/><meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"/>
<meta name="google-site-verification" content="FTeR0c8arOPKh8c5DYh_9uu98_zJbaWw5
3J-Sch9MTg"/><title>知乎 - 有问题上知乎</title><meta data-react-helmet="true" pr
operty="description" content="这是我第一次在现场看NBA，然而它是科比的最后一场比
赛。 很难装作科比占据了我的青春，虽然我在现场。 远赴洛杉矶看一场科比的退役赛，听
上去好像只有狂热粉丝才干得出- - 更何况，那张去年8月份入手时不到3000元… "/><meta
data-react-helmet="true" property="og:title" content="当我在现场告别科比，我告别
了什么？"/><meta data-react-helmet="true" property="og:url" content="https://zhu
anlan.zhihu.com/p/20751612"/><meta data-react-helmet="true" property="og:descrip
tion" content="这是我第一次在现场看NBA，然而它是科比的最后一场比赛。 很难装作科
比占据了我的青春，虽然我在现场。 远赴洛杉矶看一场科比的退役赛，听上去好像只有狂
热粉丝才干得出- - 更何况，那张去年8月份入手时不到3000元… "/><meta data-react-hel
met="true" property="og:image" content="https://pic3.zhimg.com/cf170b556aa59de65
62268a6b208fb40_b.jpg"/><meta data-react-helmet="true" property="og:type" conten
t="article"/><meta data-react-helmet="true" property="og:site_name" content="知
乎专栏"/><link data-react-helmet="true" rel="apple-touch-icon" href="
-- More --
```

这就是 GET 方式的一个最基本的运用，对于 POST 方法的实现其实和 GET 差不多，只不过多了一个 data 参数。即若添加 data 参数，就是以 POST 请求方式进行，如果没有 data 参数就是 GET 请求方式，请看下面一个 POST 案例。

```
#!env python3
from urllib import request
import urllib
data = bytes(urllib.parse.urlencode({'word': 'hello'}), encoding='utf8')
print(data)
response = request.urlopen('http://httpbin.org/post', data=data)
print(response.read().decode("utf-8"))
```

```
NORMAL | 1.py |                    python | utf-8[unix] | 14%[1,1] | 01/17-23:44
b'word=hello'
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "word": "hello"
  },
  "headers": {
    "Accept-Encoding": "identity",
    "Connection": "close",
    "Content-Length": "10",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "Python-urllib/3.6"
  },
  "json": null,
  "origin": "139.214.244.243",
  "url": "http://httpbin.org/post"
}

Press ENTER or type command to continue
```

这里通过使用 http://httpbin.org/post 网站演示（该网站可以作为练习如何使用 urllib 的一个站点使用，能够模拟各种请求操作)完成了一次 POST 请求。

通过上面的介绍我相信你对 urllib 的基础知识有了一个比较深刻的了解，但这远远不够，需要我们在实践中不断丰富我们的知识库，另外，urllib 只是爬虫一个最基础的库，务必掌握，其他一些高端的库根据需要可以自行学会。

到目前为止我们还没有进行爬一些张图片或者视频的实验。下面我们看看如何来爬一些图片。

3 爬虫小实验

在本节我将介绍如何对知乎上某个专栏的一些图片进行爬虫。

话不多说，直接上这个小实验的代码（写在 pachong.py 文件中）如下：

```
from urllib import request
from bs4 import BeautifulSoup
import re
```

```
import time
url = "https://zhuanlan.zhihu.com/p/20751612"
html = request.urlopen(url).read().decode("utf-8")
soup = BeautifulSoup(html,"html.parser")
links = soup.find_all("img","origin_image zh-lightbox-thumb",src =
re.compile(r'.jpg$'))
path = r"/home/tangxingwang/paichong_picture"
for link in links:
    print(link.attrs['src'])
    request.urlretrieve(link.attrs["src"],path+'\%s.jpg' % time.time())
```

在本实例中，我们用 Beautiful Soup 结合正则表达式的方式来提取符合要求的链接，链接要求是在 img 标签中，class=origin_image zh-lightbox-thumb，而且链接是.jpg 结尾。


```
soup = BeautifulSoup(html,"html.parser")
links = soup.find_all("img","origin_image zh-lightbox-thumb",src =
re.compile(r'.jpg$'))
```

提取出所有链接后，使用 request.urlretrieve 来将所有链接保存到本地
在终端执行上面程序代码即可爬取图片

```
python3   pachong.py
```

```
tangxingwang@tangxingwang-Lenovo-Y430P:~$ vim pachong.py
tangxingwang@tangxingwang-Lenovo-Y430P:~$ python3 pachong.py
https://pic1.zhimg.com/ef7f5f5a9dc8164c84435f2b94ee1910_b.jpg
https://pic4.zhimg.com/3ccfbedd7ee2ad34a4d120ede97ed00b_b.jpg
https://pic3.zhimg.com/346b505f04142cee865cd52c5f9c3056_b.jpg
https://pic4.zhimg.com/c878566669f2b8a0830b4ff3e5d1a693_b.jpg
https://pic2.zhimg.com/537503edc2e43de0207c8425604bdf4d_b.jpg
https://pic1.zhimg.com/a71d76afc969be3a9bc2b0d07aeb9408_b.jpg
https://pic3.zhimg.com/8f22c9830b2fe87514fd1d645fd422ba_b.jpg
tangxingwang@tangxingwang-Lenovo-Y430P:~$
```

4 github 爬虫工程

接下来我们看看 github 上一些优秀的爬虫工程。

4.1 github 图片爬虫工程

在 github 上有许多优秀的图片爬虫项目，我们选择一个对百度图片爬虫的工程详细解释下。下面是这个工程项目的 github 链接：

```
https://github.com/kong36088/BaiduImageSpider
```

我们将其 clone 下来后，有下图所示的一些文件：

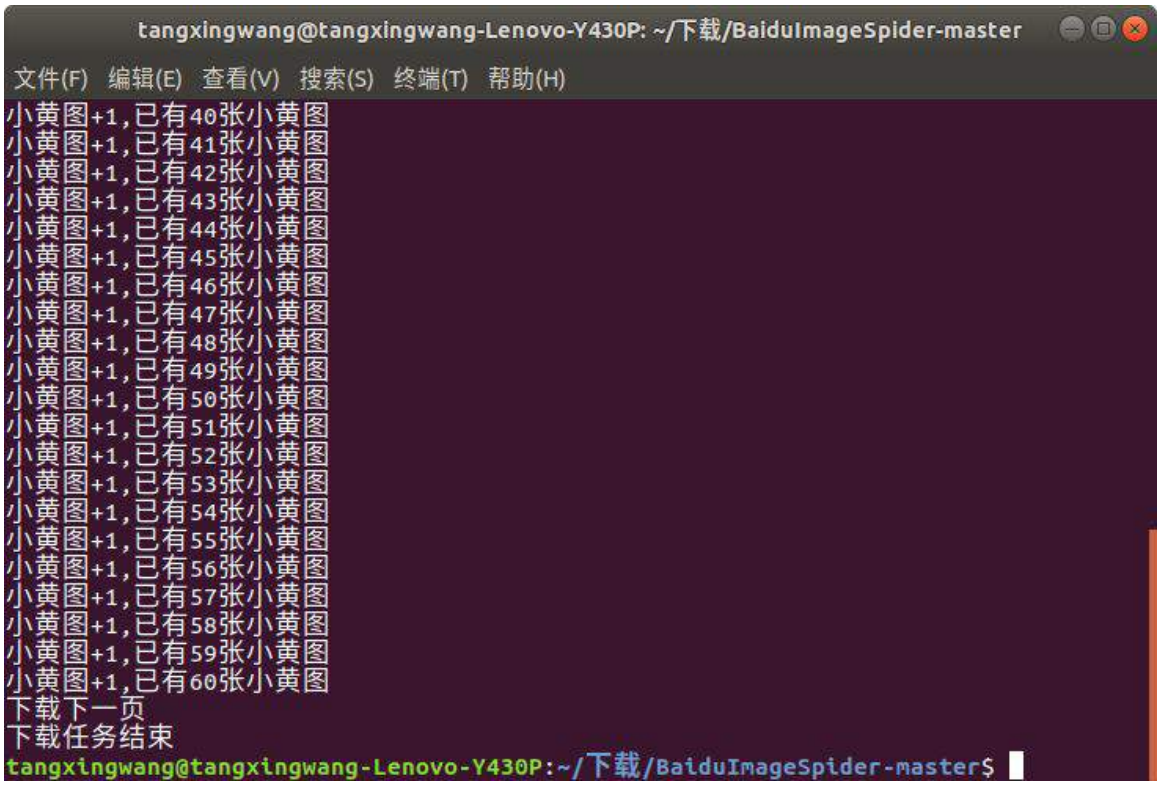
接下来我们打开 index.py 这个文件，按照自己的需求修改自己的下面这一行代码

```
if __name__ == '__main__':
    crawler = Crawler(0.05)  # 抓取延迟为 0.05

    # crawler.start('美女', 10, 2)  # 抓取关键词为“美女”，总数为 1 页（即总共 1*60=60 张），开始页码为 2
    crawler.start('科比', 1, 1)  # 抓取关键词为“科比”，总数为 1页（即总共 1*60=600 张），起始抓取的页码为 1
    # crawler.start('帅哥', 5)  # 抓取关键词为“帅哥”，总数为 5 页（即总共 5*60=300 张）
```

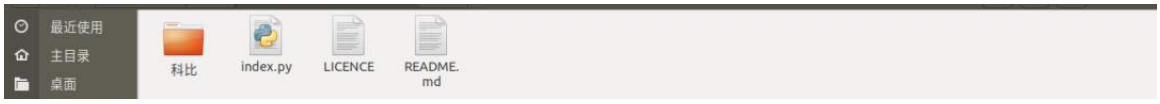
我把索引改成了科比，总数为 1 页，然后运行下面代码

```
python3 index.py
```

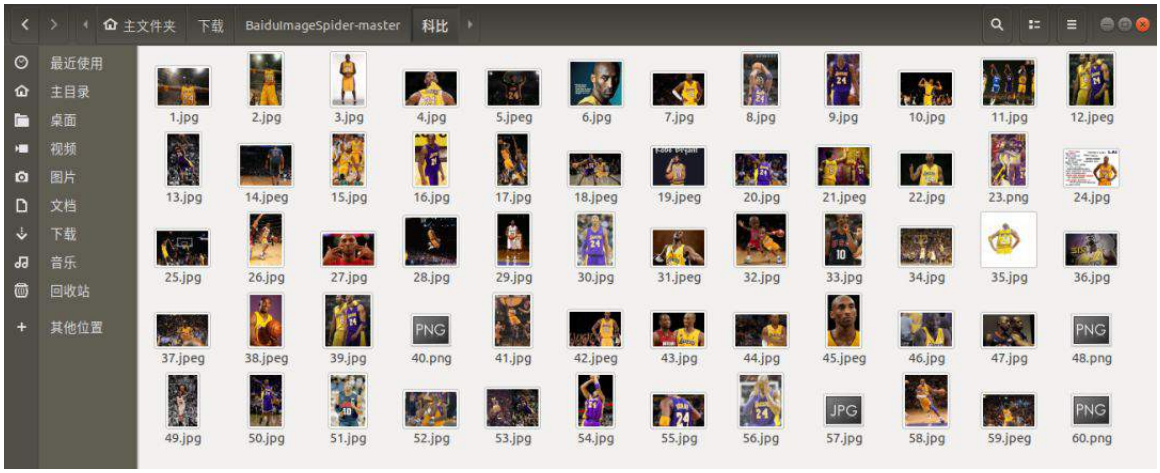
执行 python3 index.py 后你会发现有大量的图片在下载,如下图所示：



我们再看下文件的变化，你会现在的文件比我们之前 clone 下来的多了个科比文件夹，如下图所示：



打开科比这个文件夹，你会发现有许多科比的照片。



关于对百度图片爬虫就讲解到这，github 上还有大量这样的项目，如下：

1. 该 github 工程是关于对知乎里面某个问题下所有的图片进行爬虫。下面是链接：

`https://github.com/ladingwu/python_zhihu`

2. 该 github 工程是关于对微博某个用户相册里面所有的图片进行爬虫。下面是链接：

`https://github.com/lincanbin/Sina-Weibo-Album-Downloader`

3. 该 github 工程是关于对花瓣里面旅游模块图片进行爬虫，下面是链接

`https://github.com/darrenfantasy/image_crawler/tree/master/Huaban`

4. 该 github 工程是关于对 google image 进行爬虫。下面是链接：

`https://github.com/Ehco1996/Python-crawler/tree/master/Google-Image`

这就是 github 上一些关于图片爬虫的工程，当然 github 上关于图片爬虫的工程还有很多，有需要可以自己 search 再学习学习。

4.2 github 视频爬虫工程

说完图片的一些工程，我们再看看 github 上一些比较好的视频工程。下面这个链接是关
于对抖音视频进行爬虫的一个项目。链接如下：

`https://github.com/loadchange/amemv-crawler`

clone 下来有如下文件

现在打开文件`share-url.txt`,把你想要下载的抖音号分享链接编辑进去，以逗号/空格/tab/表格鍵/回车符分隔都行，可以多行。

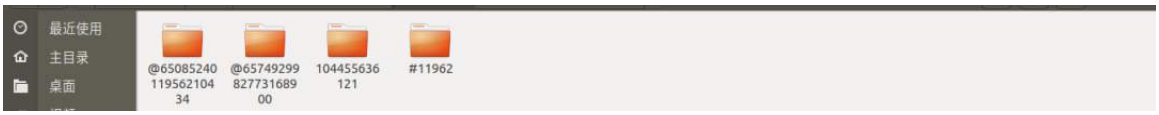
样式如下：

然后在终端执行 下面代码即可下载

`python3 amemv-video-ripper.py`

```
tangxingwang@tangxingwang-Lenovo-Y430P:~/下载/amemv-crawler-master$ python3 amem
v-video-ripper.py
Downloading v0200fca0000bgeb1obf55fct3j24ta0.mp4 from https://aweme.snssdk.com/a
weme/v1/play/?video_id=v0200fca0000bgeb1obf55fct3j24ta0&line=0&ratio=720p&media_
type=4&vr_type=0&test_cdn=None&improve_bitrate=0&iid=35628056608&device_id=46166
618999&os_api=18&app_name=aweme&channel=App%20Store&idfa=00000000-0000-0000-0000
-000000000000&device_platform=iphone&build_number=27014&vid=2ED380A7-F09C-6C9E-9
0F5-862D58F3129C&openudid=21dae85eeac1da35a69e2a0ffeaeef61c78a2e98&device_type=i
Phone8%2C2&app_version=2.7.0&version_code=2.7.0&os_version=12.0&screen_width=124
2&aid=1128&ac=WIFI.
Downloading v0200f330000bfttf7sthbi2bg3t3tu0.mp4 from https://aweme.snssdk.com/a
weme/v1/play/?video_id=v0200f330000bfttf7sthbi2bg3t3tu0&line=0&ratio=720p&media_
type=4&vr_type=0&test_cdn=None&improve_bitrate=0&iid=35628056608&device_id=46166
618999&os_api=18&app_name=aweme&channel=App%20Store&idfa=00000000-0000-0000-0000
-000000000000&device_platform=iphone&build_number=27014&vid=2ED380A7-F09C-6C9E-9
0F5-862D58F3129C&openudid=21dae85eeac1da35a69e2a0ffeaeef61c78a2e98&device_type=i
Phone8%2C2&app_version=2.7.0&version_code=2.7.0&os_version=12.0&screen_width=124
2&aid=1128&ac=WIFI.

Downloading v0300f9e0000bgl0lcdm7i9guhe3q1k0.mp4 from https://aweme.snssdk.com/a
weme/v1/play/?video_id=v0300f9e0000bgl0lcdm7i9guhe3q1k0&line=0&ratio=720p&media_
type=4&vr_type=0&test_cdn=None&improve_bitrate=0&iid=35628056608&device_id=46166
618999&os_api=18&app_name=aweme&channel=App%20Store&idfa=00000000-0000-0000-0000
```

下载后的视频保存在 download 文件件里面，里面有各个抖音号的小视频



在 github 中关于视频爬虫的工程实际上还有很多，大家可以去上面看看！

最后附上一个 github 上关于学习爬虫比较好的干货。链接如下。

`https://github.com/Ehco1996/Python-crawler`

5 总结

AI 领域必须掌握的数据爬虫基础就讲到这里，这方面的知识还有很多，大家平时还需要多注意学习！

【AI 白身境】深度学习中的数据可视化

本篇是《AI 白身境》的第八篇，所谓白身，就是什么都不会，还没有进入角色。上一篇我们已经讲述了如何用爬虫爬取数据，那爬取完数据之后就应该是进行处理了，一个很常用的手段是数据可视化。

通过数据可视化，可以更加直观地表征数据，在深度学习项目中，常需要的数据可视化操作包括原始**图片数据的可视化，损失和精度的可视化**等。

作者 | 言有三 臧小满
编辑 | 言有三 臧小满

1 什么是数据可视化？

每每提到数据可视化，大家脑中可能会浮现很各种图表、西装革履的分析师、科幻大片中酷炫的仪表。

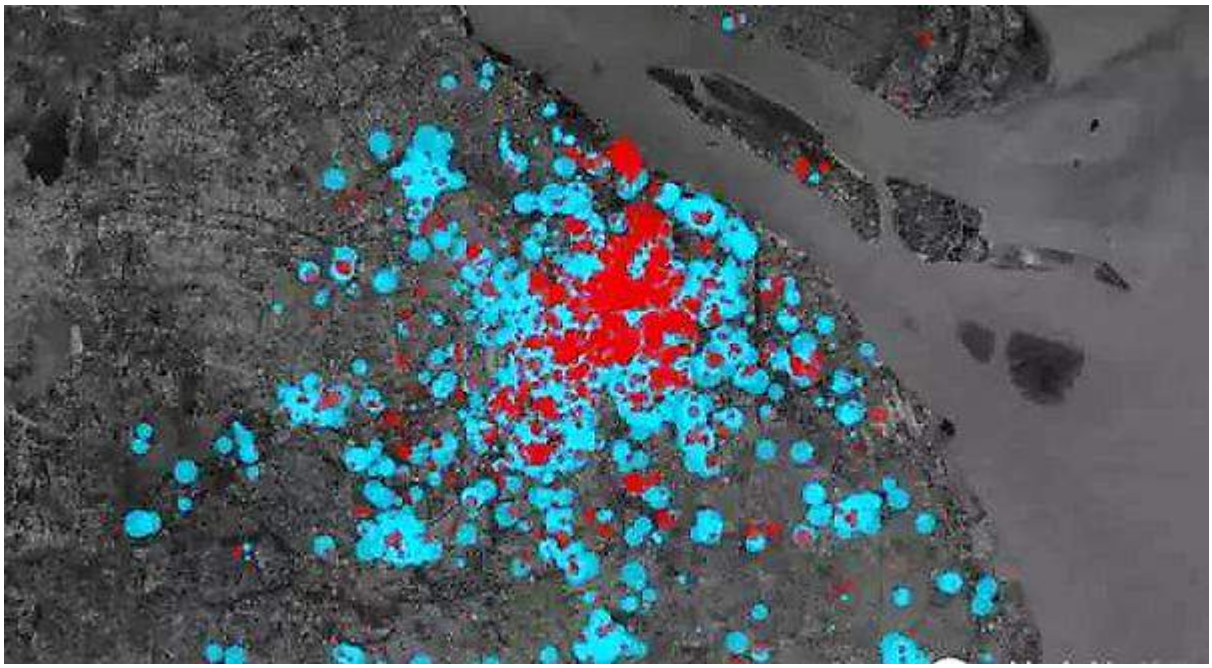
其实不用那么复杂，数据可视化早就融合进你我的生活，地铁线路图、公交时刻表，天气预报中的气象地图等都是很常见的。

为什么要进行可视化？

因为**人是视觉动物，对于图像的敏感度**要比对纯数字的敏感度高的多。

人类对图像的处理速度比文本快 6 万倍，同时人类右脑记忆图像的速度比左脑记忆抽象文字快 100 万倍。数据可视化正是利用人类天生技能来增强数据处理和组织效率。

举个简单的例子，计划买一套房产作为投资， 想要了解“去年上海房价哪里涨幅最大”，现以图作答， 把去年的增长率体现在图上，以 20%作为分界， 增长超过 20%的标红色， 超过越多则越大， 不足的标记成蓝色， 如下图，可以很快 get 到哪个区域的大幅度涨幅。



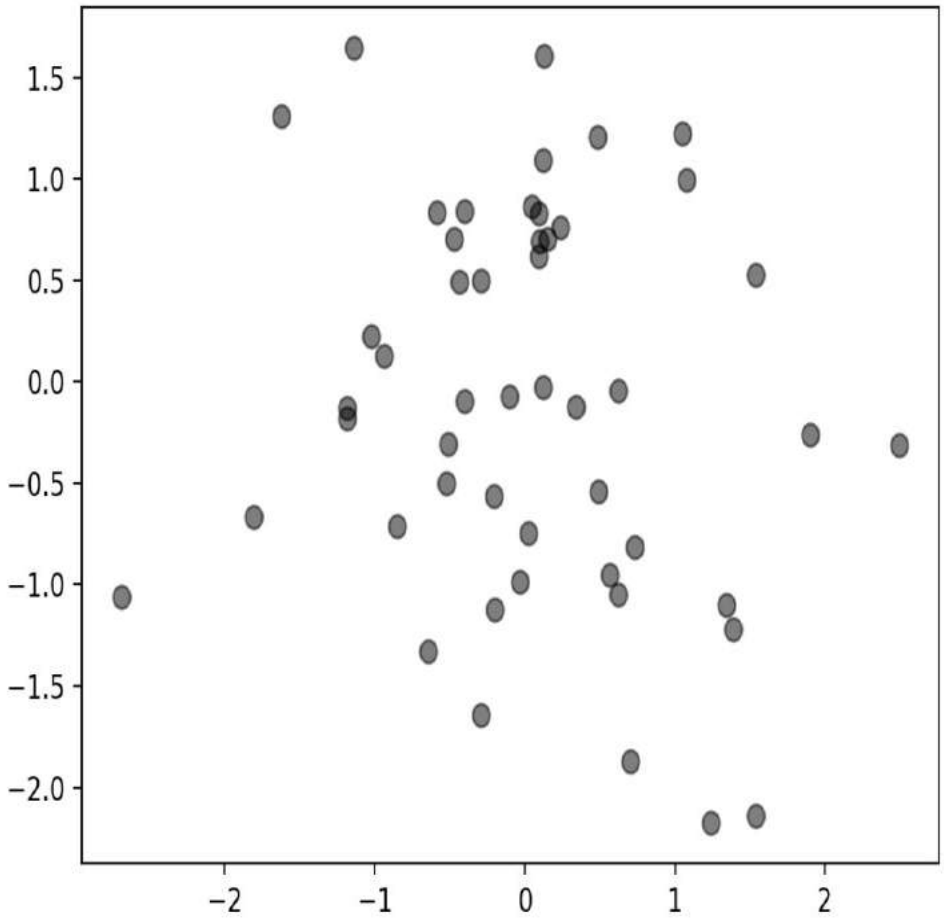
可视化将数字抽象成了更方便我们观察和感受的图表，因此需要熟悉使用。

2 低维数据可视化

数据有不同的维度，我们最常接触的就是一维，二维的数据，在机器学习任务中，包括损失函数等统计指标。

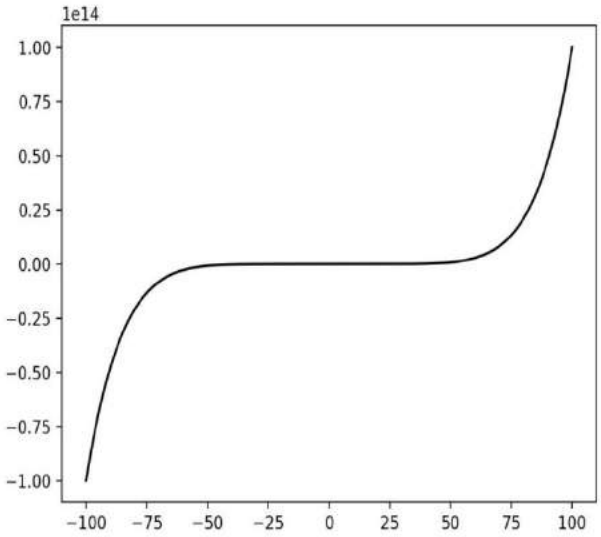
2.1 散点图

散点图，常用于分析离散数据的分布。比如我们有一个数据集，里面的图片有不同的大小，我们可以利用 x，y 轴分别对应图片的宽高，从而画出图片尺度的空间分布情况。越密集的地方，说明该尺度类型的图越多，如下图所示。



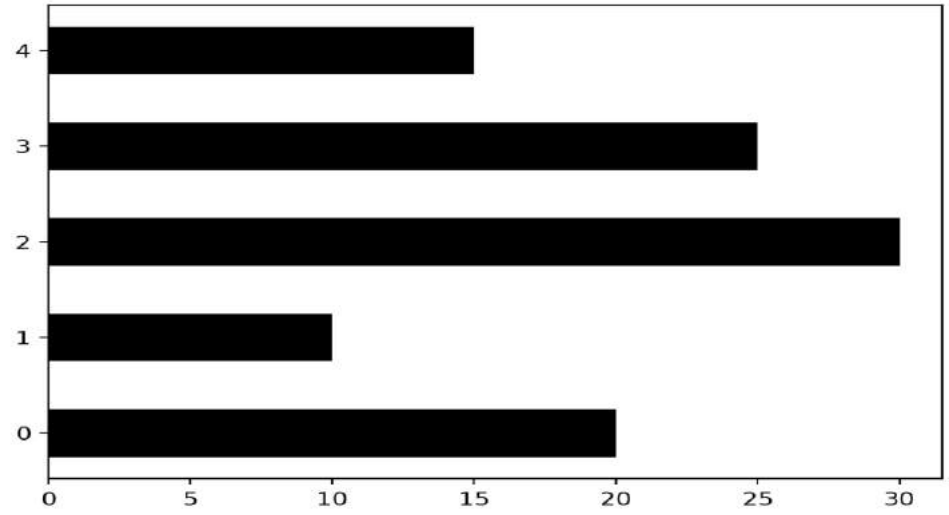
2.2 折线图

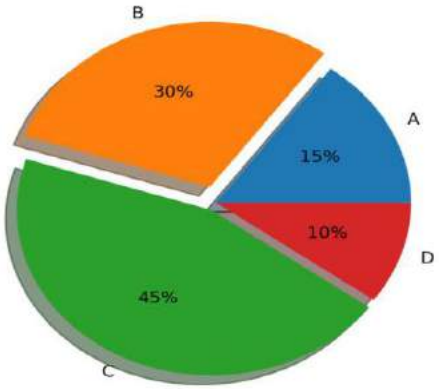
折线图是用于分析变量随另一个变量的变化关系，我们平常接触最多的 loss 曲线图，accuracy 曲线图就是这一种，可以看指标随着训练过程的变化判断收敛情况，从而推测模型训练的好坏，折线图被广泛应用于各类分析，如下图所示。



2.3 直方图，饼状图

这两种图，都常用于统计数据的分布比例以及响应幅度，比如一幅图片的亮度分布情况，不同网络层的参数量，计算时间代价。





这几种图，适合对有时序变化的一维向量，有统计分布的一维向量，或者二维图像的尺度等信息进行可视化。

3 高维数据可视化

在机器学习任务中，数据通常是用成百上千维的向量表示，而超过 3 维的向量，就已经超过了人类的可视化认知，因此通常需要对数据进行降维。

数据降维方法可以分为**线性方法和非线性方法**。其中线性方法包括 **PCA 和 LDA**，而非线性方法有保留局部特征、基于全局特征等方法，以 **t-SNE** 为代表。下面我们主要介绍 PCA 和 t-SNE 方法。

3.1 PCA 降维

PCA，全称是 Principal components analysis，这是一种分析、简化数据集的技术。PCA 常用于减少数据集的维数，同时保持数据集中对方差贡献最大的特征，原理是**保留低阶主成分，忽略高阶主成分，因为低阶成分保留了数据最多的信息**。

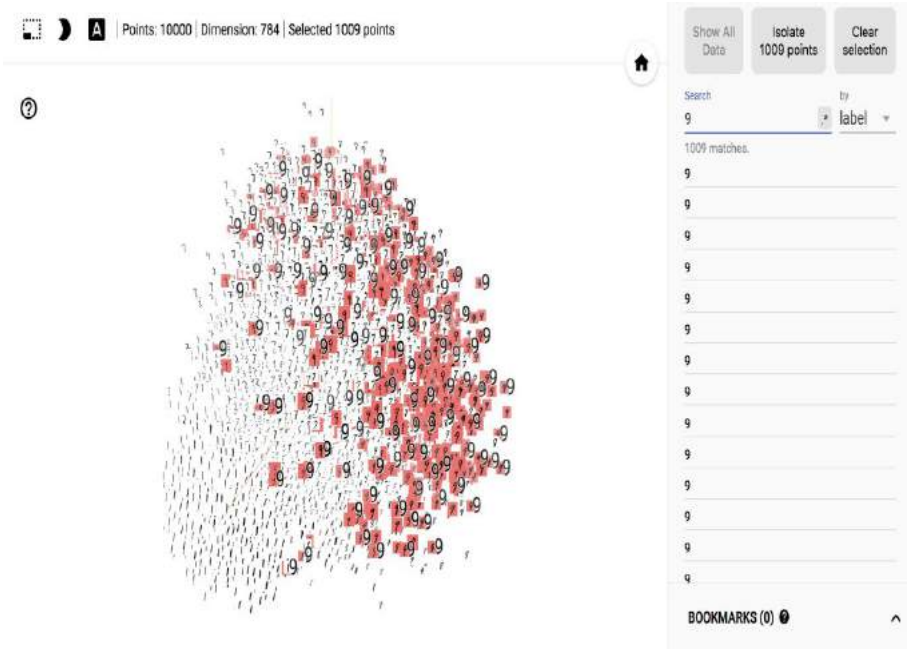
假定 X 是原始数据，Y 是降维后的数据，W 是变换矩阵，Y=XW。假如我们需要降到 3 维以便于我们可视化，那就取 Y 的前三个主成分作为原始属性 X 的代表。

我们采用 Google 开源的网页版数据可视化工具 Embedding Projector 来进行可视化，链接如下：

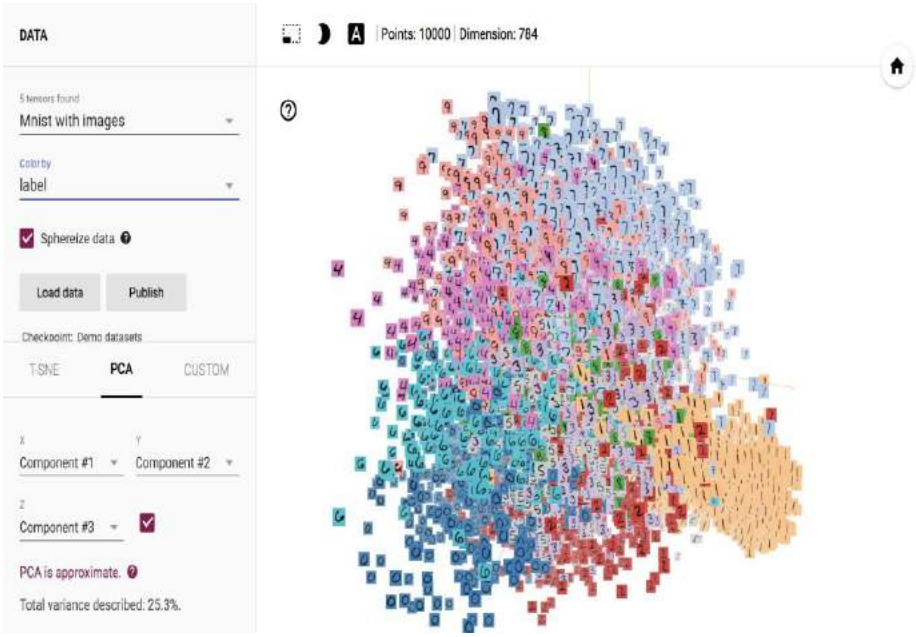
http://projector.tensorflow.org/

选择 MNIST 作为可视化例子，它的原始维度为 10000×784，即 10000 张 28×28 的图像。

利用这个工具我们进行 PCA 的可视化，降低到 3 个维度后，我们可以选择某个数字进行可视化。下图就是数字 9 的分布，可以看到，总共有 1009 个样本，数据的分布在物理空间上具有一定的聚类特性。



还可以用不同的颜色查看全体数据的分布，从这里可以更好的看出不同类的分布规律。



3.2 t-SNE 降维

SNE 全称是 Stochastic Neighbor Embedding，它将**数据点之间高维的欧氏距离转换为表示相似度的条件概率**，目标是将高维数据映射到低维后，尽量保持数据点之间的空间结构，从而那些在高维空间里距离较远的点，在低维空间中依然保持较远的距离。

t-SNE 即 t-distributed stochastic neighbor embedding，t-SNE 用联合概率分布替代了 SNE 中的条件概率分布，解决了 SNE 的不对称问题。通过引入 t 分布，解决了同类别之间簇的拥挤问题。

t-SNE 方法实质上是一种聚类的方法，对于一个空间中的点，周围的其他点都是它的“邻居”，方法就是要试图使所有点具有相同数量的“邻居”。

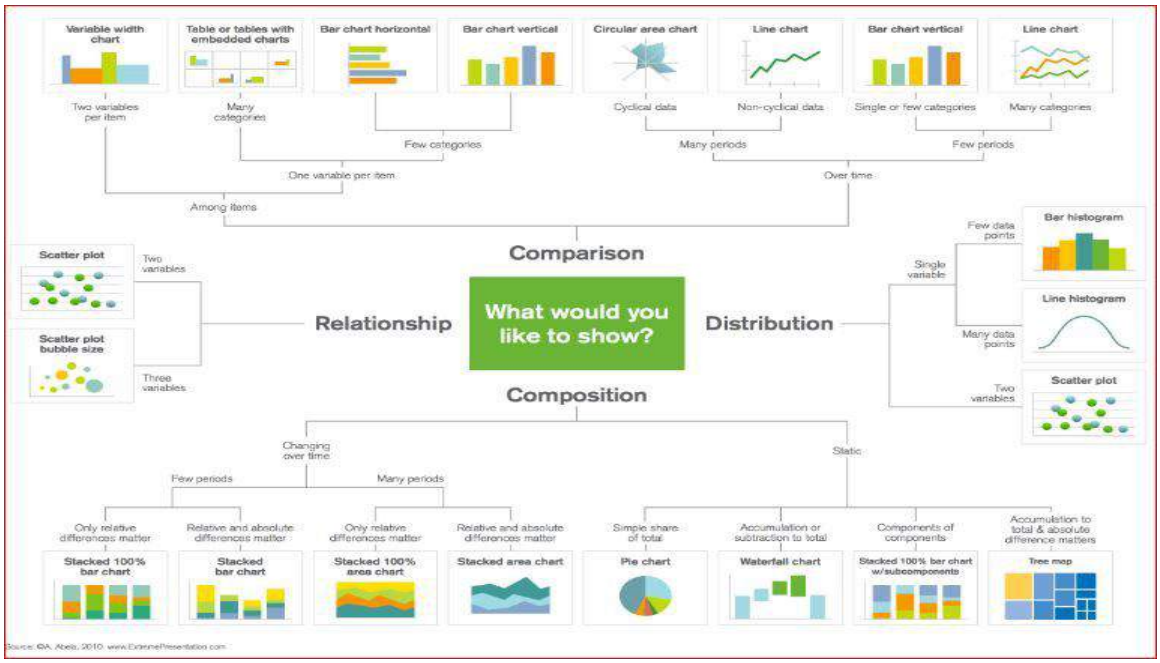
t-SNE 经过学习收敛后，通过投影到 2 维或者 3 维的空间中可以判断一个数据集有没有很好的可分性，即是否同类之间间隔小，异类之间间隔大。如果在低维空间中具有可分性，则数据是可分的，如果不具有可分性，可能是数据不可分，也可能仅仅是因为不能投影到低维空间。

下图是 t-SNE 可视化结果图，可以看出，数字都有很明显的聚类效果。

在进行一个机器学习任务之前，通过可视化来对数据集进行更深刻的认识，有助于预估任务的难度，在遇到困难后也会更加容易找到解决方案。

4 python 数据可视化项目

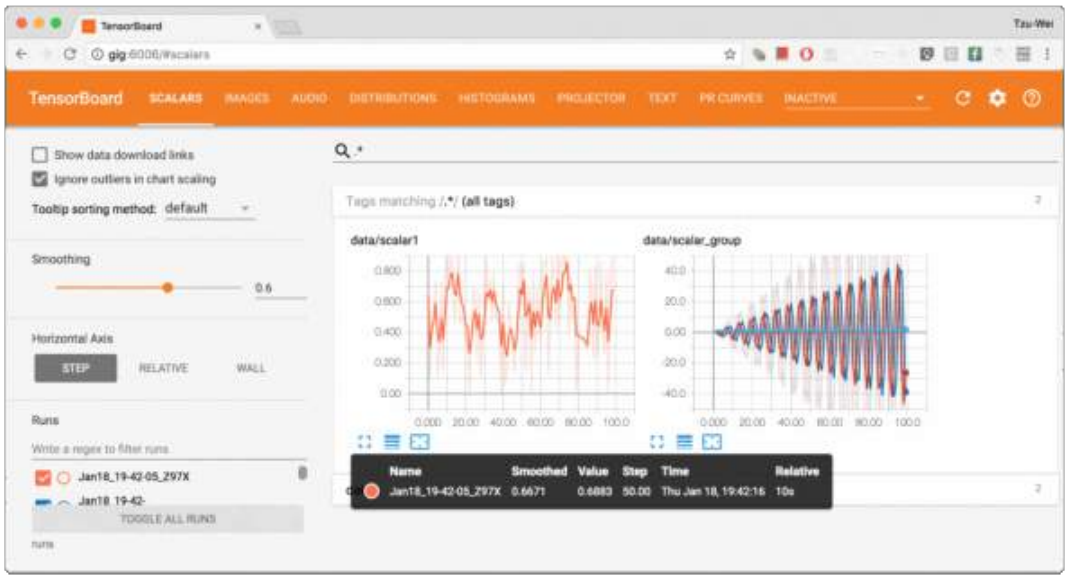
考虑到 python 是第一大机器学习编程语言，同时开源项目居多，所以我们只关心 python 相关的工具，而且 python 也基本可以满足需求。



可视化的项目太多了，下面基于 python 和 GitHub 的数据，随便推荐几款。

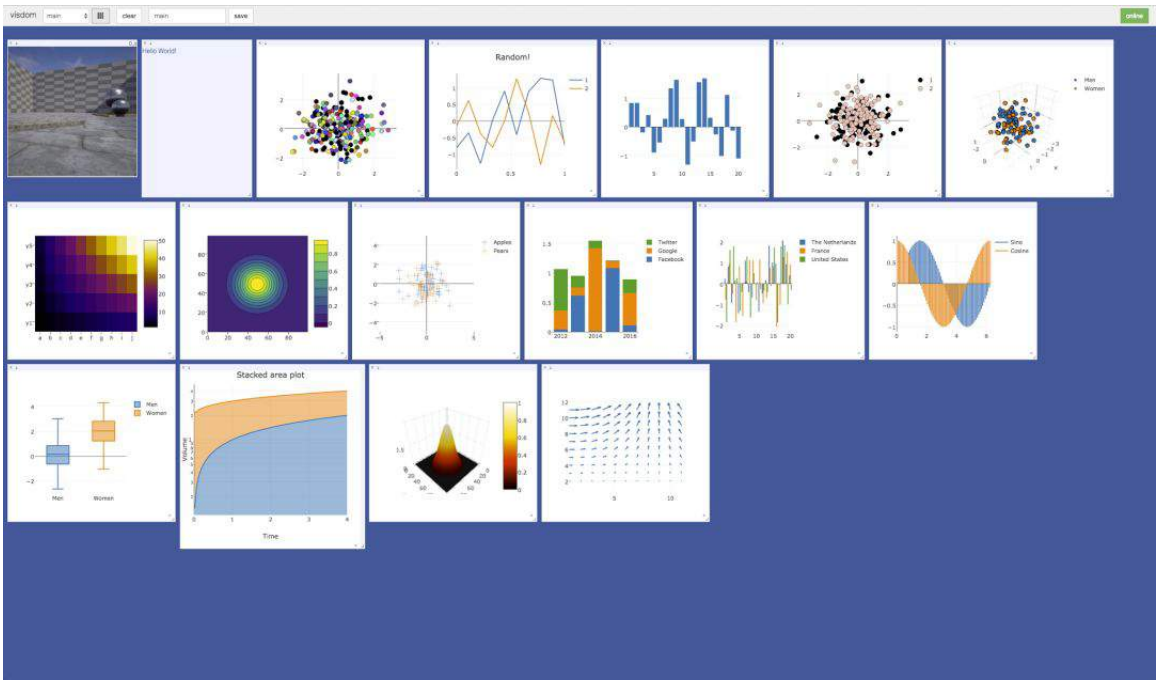
1. tensorboard 和 tensorboardX，想必不需要多做介绍，后者大家可能不熟悉，被开发用来支持 chainer, mxnet, numpy，4000+star。

https://github.com/lanpa/tensorboardX



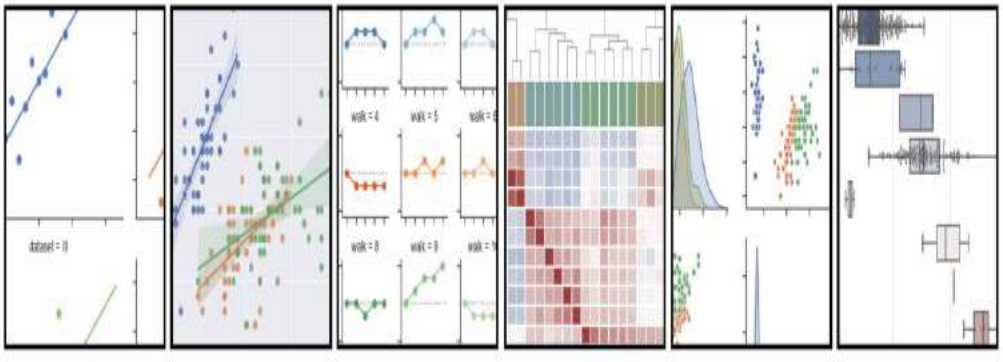
2. visdom，支持 numpy 和 torch 的工具，常用于 pytorch 数据可视化，很强大，5000+star。

https://github.com/facebookresearch/visdom



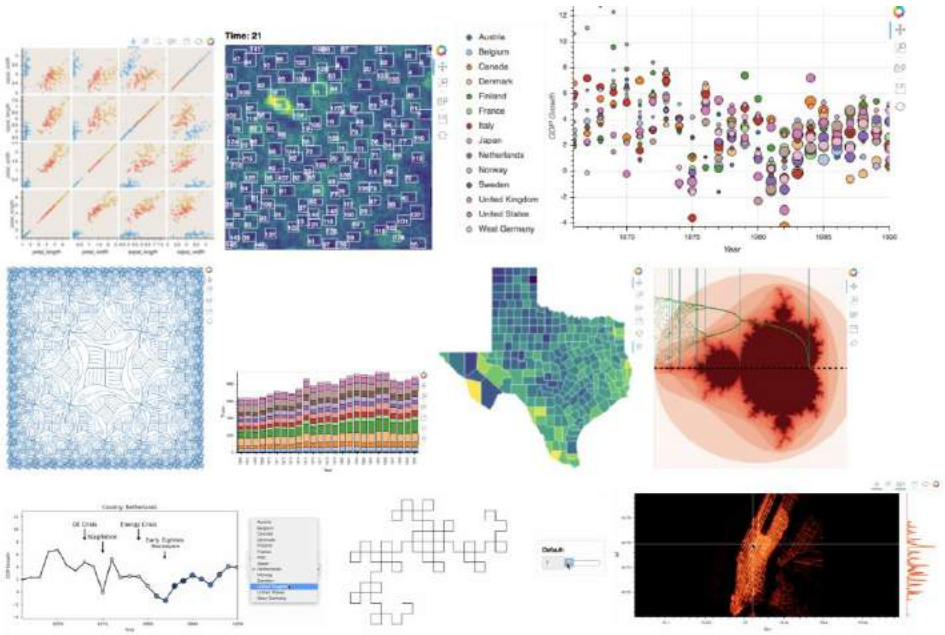
3. seaborn：一款基于 matplotlib 的工具，简单来说，就是有更高的 API，画出的图也好看，5000+star，主要处理低维数据。

https://github.com/mwaskom/seaborn



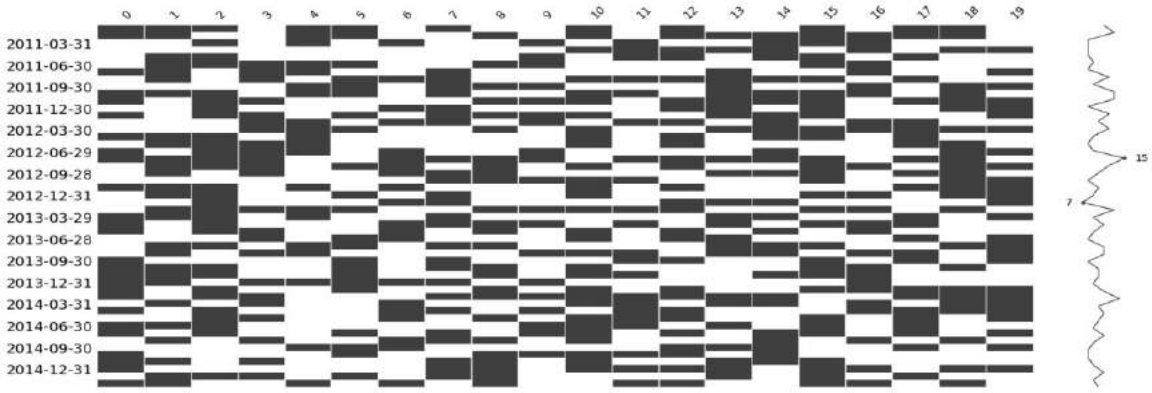
4. holoviews：很酷炫的工具，与 season 差不多，1000+star。

https://github.com/ioam/holoviews



5. missingno：一款缺失数据可视化工具，非常适合分析数据集的完整性，1000+star。

https://github.com/ResidentMario/missingno



就这么多，以后再集中讲可视化工具。

6 总结

数据可视化抽象了数据本身真正的价值，熟练掌握可视化对于分析数据的特征和深度学习模型的性能是必要的技能。

【AI 白身境】入行 AI 需要什么数学基础：左手矩阵论，右手微积分

本篇是新专栏《AI 白身境》的第九篇，所谓白身，就是什么都不会，还没有进入角色。咱们这个系列接近尾声了，今天来讲一个非常重要的话题，也是很多的小伙伴们关心的问题。要从事 AI 行业，吃这碗饭，至少应该先储备一些什么样的数学基础再开始。

下面从**线性代数，概率论与统计学，微积分和最优化 3 个方向说起**，配合简单案例，希望给大家做一个抛砖引玉，看完之后能够真正花时间去系统性补全各个方向的知识，笔者也还在努力。

作者 | 言有三

1 线性代数

1.1 向量

什么是数学？顾名思义，一门研究“数”的学问。学术点说，线性代数是一个数学分支，来源于希腊语μαθηματικ??（mathematikós），意思是**“学问的基础”**。

数学不好，就不要谈学问了，只能算知识（自己瞎加的，欢迎喷）。

按照维基百科定义：数学是利用符号语言研究数量、结构、变化以及空间等概念的一门学科，从某种角度看属于形式科学的一种。所以一看见数学，我们就想起符号，方程式，简单点比如这个。

$$y = x^2$$

复杂的比如这个

$$-\frac{\hbar^2}{2m}\nabla^2\Psi(\mathbf{r},t)+V(\mathbf{r})\Psi(\mathbf{r},t)=i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},t)$$

代数是数学的一个分支，它的研究对象是向量，涵盖线、面和子空间，起源于对二维和三维直角坐标系的研究。

我们都知道欧式空间，任何一个向量(x,y,z)可以由三个方向的基组成

$$(x,y,z) = x(1,0,0)+y(0,1,0)+z(0,0,1)$$

它的维度是 3，拓展至 n 就成为 n 维空间，这 N 维，相互是独立的，也就是任何一个都不能由其他的几维生成，这叫**线性无关**，很重要。

2.2 线性回归问题

用向量表示问题有什么用呢？

假如基友今天约你去吃饭，没有说好谁买单，而根据之前的惯例你们从来不 AA，今天你刚交了房租，没钱了，那么该不该去呢？我们可以先回归一下他主动买单的概率，先看一下和哪些变量有关，把它串成向量。

> X=(刚发工资，刚交女朋友，刚分手，要离开北京，有事要我帮忙，无聊了，过生日，就是想请我吃饭，炒比特币赚了，炒比特币亏了，想蹭饭吃)，共 11 维，结果用 Y 表示

Y=1，表示朋友付款，Y=-1，表示不付款

好，我们再来分析下：

> 和 Y=1 正相关的维度： 要离开北京，有事要我帮忙，过生日，就是想请我吃饭，炒比特币赚了
>
> 和 Y=-1 正相关的维度：想蹭饭吃
>
> 暂时关系不明朗的维度：刚发工资，刚交女朋友，刚分手，无聊了，炒比特币亏了

好，拿出纸笔，今天是 2019 年 1 月 22 日，据我所知，这货就是一个典型的死宅摩羯工作狂

> 刚发工资=0，时候没到
> 刚交女朋友=0，不可能
> 刚分手=0，没得选
> 要离开北京=0，不像
> 有事要我帮忙=0，我能帮上什么忙
> 无聊了=1，估计是
> 过生日=0，不对
> 就是想请我吃饭=0，不可能
> 炒比特币赚了=？，不知道
> 炒比特币亏了=？，不知道
> 想蹭饭吃=？，不知道

这下麻烦了，有这么多选项未知，假如我们用一个权重矩阵来分析，即 y=WX，W 是行向量，X 是列向量

$$W^T = w_1, \cdots w_n$$

$$X = \begin{matrix} x_1 \\ \cdots \\ x_n \end{matrix}$$

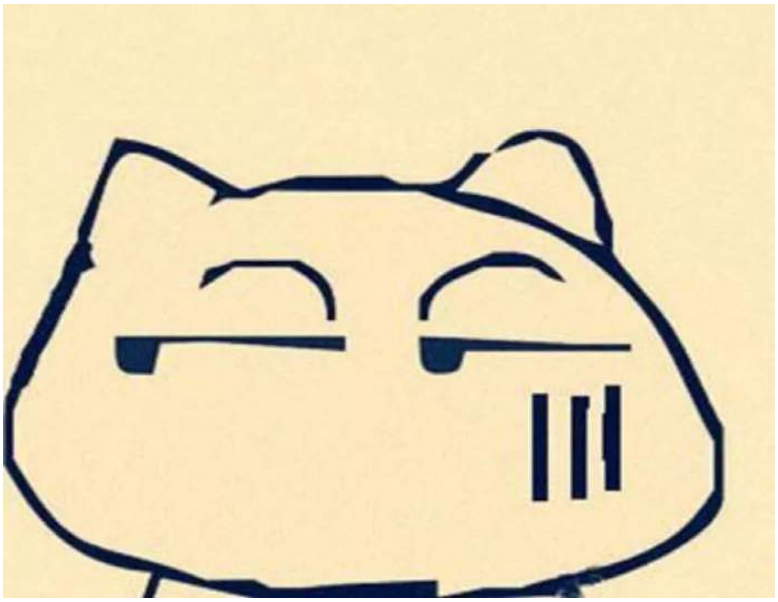
X1 到 xn 就是前面那些维度。现在等于

$$(0,0,0,0,0,1,0,0,\alpha,\beta,\gamma),\quad \alpha,\beta,\gamma \in (0,1)$$

假如我们不学习参数，令所有的 wi 与 y=1 正相关的系数为 1，与 y=-1 正相关为-1，关系不明的随机置为 0.001 和-0.001，那么就有下面的式子

$$y = (0.001,0.001,-0.001,1,1,-0.001,1,1,1,0.001,-1) \times (0,0,0,0,0,1,0,0,\alpha,\beta,\gamma)^T = \alpha + 0.001 \times \beta - \gamma$$

还是 3 个未知数，问题并没有得到解决。



不过我们还是可以得到一些东西：

- 我们的模型还没有得到训练，现在的权重是手工设定的，这是不合理的，应该先抓比如 1 万个样本来填一下报告，把 X 和 Y 都填上。当然，要保证准确性，不能在报告中填了说自己会请客（y=1），实际吃起来就呵呵呵??。这样就是标签打错了，肯定学不到东西。
- 从 X 来看，这个朋友还是可以的，与 y=1 正相关的变量更多，但是，未必！因为现在 X 的维度太低了，比如这个朋友是不是本来就是小气鬼或者本来就喜欢请人吃饭，比如是来我家附近吃还是他家附近吃，比如他吃饭带不带女孩等等。
- 上面提到了一些随机性，比如权重 W 的随机性，0.001 或者-0.001，X 本身的噪声 α，β，γ。

是不是很复杂，现实问题本来就很复杂嘛。不过如果你没有经济问题，那就可以简单点，不管这个模型，只问你今天想不想吃饭，是就去，不想吃就不去。

线性代数就说这么多，后面想好好学，一定要好好修行线性代数和矩阵分析，咱们以后再说，书单如下。

以下是一些关键词，如果都熟练了解了第一阶段也就 OK 了。

标量，向量，特征向量，张量，点积，叉积，线性回归，矩阵，秩，线性无关与线性相关，范数，　奇异值分解，行列式，主成分分析，欧氏空间，希尔伯特空间。

2 概率论与统计学

2.1 概率论

概率大家都知道吧，研究的是随机性事件。大家应该都曾经饱受**贝叶斯公式**的折磨。

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

概率论中有以下几个概念，还是以之前的吃饭问题，朋友主动叫我吃饭为事件 X，也叫观测数据，他请客了事件为 Y，有以下几个概率，其中 P(A|B)是指在事件 B 发生的情况下事件 A 发生的概率。

(1) X 的先验概率，即朋友主动喊我吃饭的概率 p(X)，与 Y 无关。
(2) Y 的先验概率 p(Y)：即单纯的统计以往所有吃饭时朋友请客的概率 p(Y)，与 X 无关。
(3) 后验概率 p(Y|X)：就是给出观测数据 X 所得到的条件概率，即朋友喊我吃饭，并且会请客的概率。

anyway，饭我们吃完了，现在回家，结果未来的女朋友打来电话问去干嘛了，气氛有点严肃，原来是吹牛皮过程中没有看微信漏掉了很多信息。只好说去应酬了，妹子不满意问你还有钱吃饭，谁请客。我说不吃白不吃啊，朋友请。

妹子又问，谁主动提出吃饭的！

正好，那不就是要算后验概率 p(X|Y)吗？也就是饭吃了，谁提议的。

于是故作聪明让妹子猜，还给了一个提示可以用贝叶斯公式，并且已知 p(Y)=0.2，p(X)=0.8，再加上上面算出来的 p(Y|X)

$$p(X|Y) = \frac{P(X)P(Y|X)}{P(Y)} = 0.8 \times \frac{\alpha + 0.001 \times \beta - \gamma}{0.2} = 4(\alpha + 0.001 \times \beta - \gamma)$$

好了又回到了这个问题，3 个未知变量。

不过没关系，我们可以先用它们的数学期望来替换掉，数学期望就是一个平均统计。

$$E(\alpha)=0.5,\ E(\beta)=0.5,\ E(\gamma)=0.5$$

$$E\left(p(X|Y)\right)=E(4(\alpha+0.001\times\beta-\gamma))=4\times0.001\times0.5=0.002$$

这说明什么？说明这一次吃饭，是朋友先动的嘴的概率 p(X|Y)=0.002，那么今天 99.8%是自己跑出去蹭吃吹牛皮了。

接下来的问题就是搓衣板是跪还是不跪，贝叶斯公式解决不了。

事情结束后，要想好好搞下去，肯定是要学好概率论和统计学习的。

同样，有一些关键词要掌握。

不确定性，随机变量，大数定律，联合分布，边缘分布，条件概率，贝叶斯公式，概率密度，熵与交叉熵，期望，最大似然估计，正态分布/高斯分布，伯努利分布，泊松分布，概率论与统计推断，马尔可夫链，判别模型，生成模型。

有意思的是：概率论还有一些东西是有点违背认知的，比如生日悖论。

一个班上如果有 23 个人，那么至少有两个人的生日是在同一天的概率要大于 50%，对于 60 或者更多的人，这种概率要大于 99%。大家都是上过学的少年，你在班上遇到过同一天生日的吗？

2.2 传统机器学习算法基础

传统机器学习算法本来不应该放在这里说，但是因为其中有一部分算法用到了概率论，所以也提一句。

有很多人在知乎上问，**搞深度学习还需要传统机器学习基础吗？**当然要！且不说这个传统机器学习算法仍然在大量使用，光是因为它经典，就值得学习一下，依旧推荐一本书。

机器学习完成的任务就是一个模式识别任务，**机器学习和模式识别**这两个概念实际上等价，只是历史原因说法不同。

一个模式识别任务就是类似于识别这个图是不是猫，这封邮件是不是垃圾邮件，这个人脸是不是你本人之类的高级的任务。

传统的机器学习算法有两大模型，**一个是判别模型，一个是生成模型**，我们以前讲过，大家可以去看。

【技术综述】有三说 GANs（上）

传统机器学习算法就不展开了，太多。

3 微积分与最优化

3.1 导数

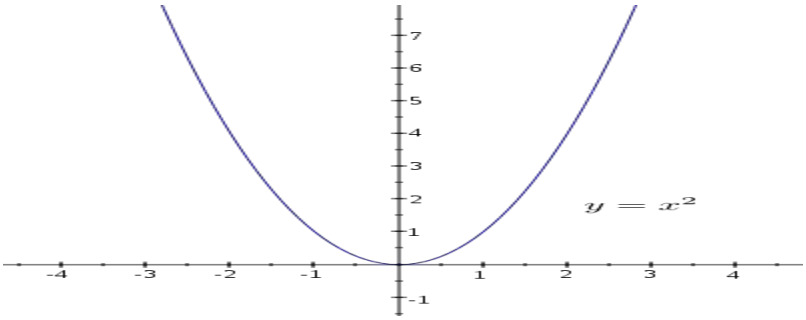
机器学习就是要学出一个模型，得到参数嘛，本质上就是优化一个数学方程，而且通常是离散的问题，这个时候大杀器就是微积分了。

微积分是什么，根据维基百科：

微积分学（Calculus，拉丁语意为计数用的小石头）曾经指无穷小的计算，就是一门**研究变化**的学问，更学术点说就是研究**函数的局部变化率**，如下。

$$Y = X^2, \frac{\partial Y}{\partial X} = 2X$$

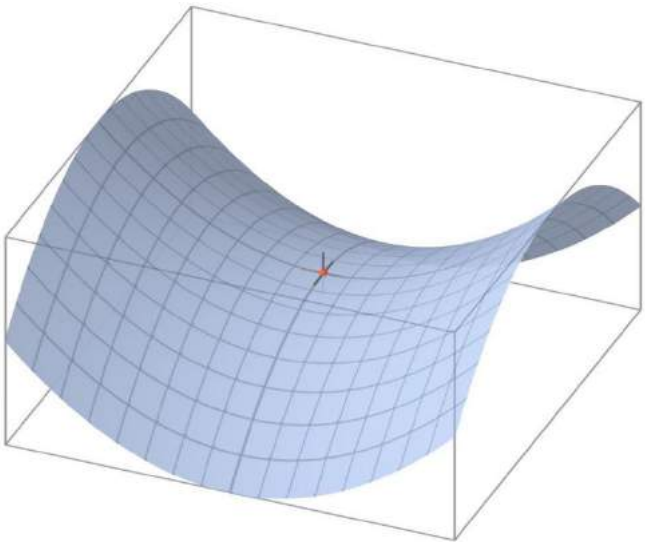
可知，在不同的 X 处它的导数是不相等的。如果遇到了一个导数为 0 的点，它很有可能就是最大值或者最小值，如下面的 x=0 点取得最小值 y=0。



导数反映了 y 的变化趋势，比如这个方程 x>0 时，导数大于 0，则 y 随着 x 的增加而增加。x<0 时，导数小于 0，则 y 随着 x 的增加而减小。

所以看导数，我们就得到了目标 y 的变化趋势，而深度学习或者说机器学习中需要优化的目标就是一个 Y，也称之为**目标函数，价值函数，损失函数**等等。通常我们定义好一个目标函数，当它达到极大值或者极小值就实现了我们的期望。

不过还有个问题，就是导数等于 0，一定是极值点吗？未必，比如鞍点。



上面的小红点就是鞍点，在这个曲面上，它在某些方向的导数等于 0，但是显然它不是极值点，不是极大也不是极小，正因如此，给后面的优化埋下了一个坑。

如果你真的微积分也忘了，就需要补了。

3.2 数值微分

前面说了，机器学习就是要求解目标的极值，极大值极小值是等价的不需要纠结，通常我们求**极小值**。

上面的函数我们轻轻松松就求解出了导数，从而得到了唯一的极值，这叫做**解析解**，答案很唯一，用数学方程就能手算出来。

但是实际要优化的神经网络上百万个参数，是不可能求出解析解的，只能求**数值近似解，就是用数值微分的方法去逼近。**

数值微分的核心思想就是**用离散方法近似计算函数的导数值或偏导数值**，相信同学们在课程中都学过。

向前差商公式：

$$f(x)' = \frac{f(x+h) - f(x)}{h}$$

向后差商公式：

$$f(x)' = \frac{f(x) - f(x-h)}{h}$$

中心差商公式：

$$f(x)' = \frac{f(x+h) - f(x-h)}{2h}$$

有了感觉咱们接着说。

那么，一般情况下要求解任意函数极值的方法是什么呢？在深度学习中就是**梯度下降法**。

梯度下降法可以说是最广泛使用的最优化方法，在目标函数是凸函数的时候可以得到全局解。虽然神经网络的优化函数通常都不会是凸函数，但是它仍然可以取得不错的结果。

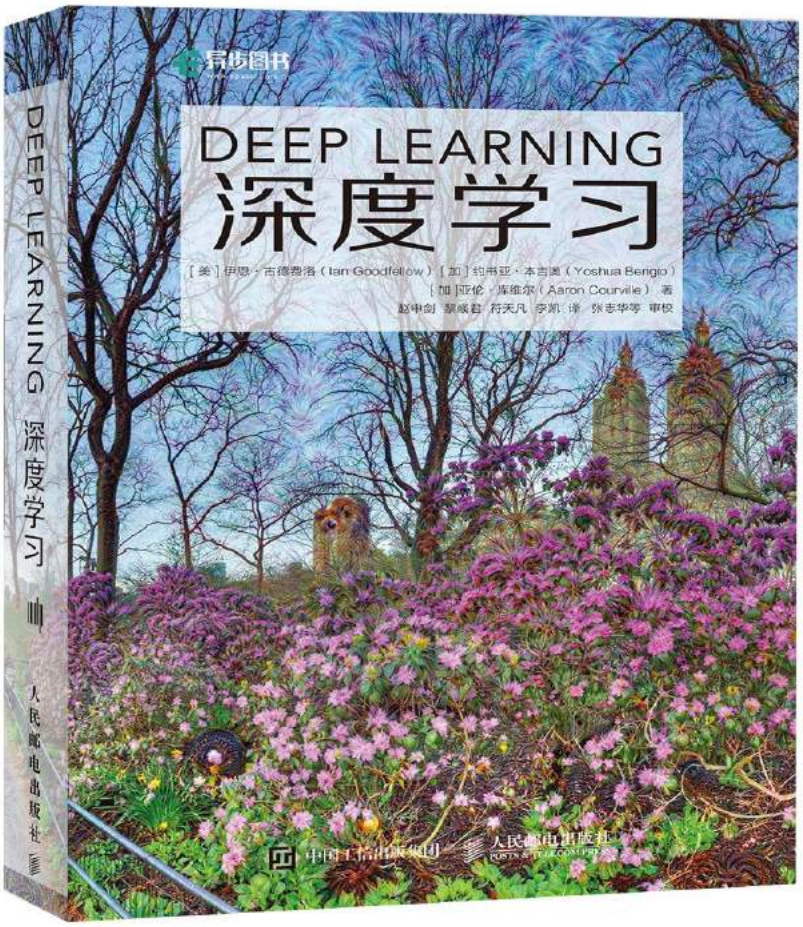
梯度下降法的核心思想就是(公式比较多，就截图了)：

一个函数$f(x)$，记$f'(x)$为它的梯度，对于足够小的ε，$f(x-\varepsilon sign(f'(x)))$是小于$f(x)$的。比如$y=x^2$这个函数，取$\varepsilon=0.001$，当$x>0$时，$f'(x)>0$，$f(x)$是单调递增函数，$x-\varepsilon sign(f'(x))<x$，所以$f(x-\varepsilon sign(f'(x)))$当然就小于$f(x)$。当$x<0$时，$f'(x)<0$，$f(x)$是单调递减函数，$x-\varepsilon sign(f'(x))>x$，所以$f(x-\varepsilon sign(f'(x)))$当然也小于$f(x)$。

这一套对所有的函数 f(x)都通用，所以以导数的反方向进行搜索，就能够减小 f(x)，而且这个方向还是减小 f(x)的最快的方向，这就是所谓的梯度下降法，也被称为“最速下降法”，参数更新方法如下。

$\theta_{n+1} = \theta_n - l\nabla_\theta J(\theta)$，θ即参数，∇是导数，J是优化目标，l是学习率

关于微积分和最优化，咱们就点到为止了，不然就超出了白身境系列的要求。要补这方面的知识，就比较多了，建议先找花书中的对应章节看看，找找感觉再说。



最优化的方法还有很多，目前在神经网络优化中常用的是一阶优化方法，不过二阶优化方法也慢慢被研究起来，最后还是给出一些关键词去掌握。

导数，偏导数，线性规划，二次规划，动态规划，Hessian matrix，损失函数，正则项，一阶优化方法(梯度下降法)，二阶优化方法(牛顿法)等等。

好嘞，掌握了这些，就大胆往前走，不用怕了。

4 总结

数学这种东西，学习就是三步曲，一看书，二做题，三应用，其他学习方法比如看视频听课基本都是扯淡。

另外，数学怎么好都不过分。希望小白看完还能爱数学，毕竟这才刚刚开始一点点。

【AI 白身境】一文览尽计算机视觉研究方向

本篇是《AI 白身境》的第 10 篇，所谓白身，就是什么都不会，还没有进入角色。相信看了前面的几篇文章后很多朋友已经等不及快速入行了，本文就来介绍一下**计算机视觉的各大研究方向及其特点**。所谓计算机视觉，即 compute vision，就是通过用计算机来模拟人的视觉工作原理，来获取和完成一系列图像信息处理的机器。计算机视觉属于机器学习在视觉领域的应用，是一个多学科交叉的研究领域，涉及数学，物理，生物，计算机工程等多个学科，由此也可以想象到计算机视觉的研究范围非常广，也是图像，语音，自然语言处理领域中从业人数最多的。

作者 | 言有三
编辑 | 言有三

1 图像分类

1.1 基本概念

图像分类是计算机视觉中最基础的一个任务，也是几乎所有的基准模型进行比较的任务，从最开始比较简单的 10 分类的灰度图像手写数字识别 mnist，到后来更大一点的 10 分类的 cifar10 和 100 分类的 cifar100，到后来的 imagenet，图像分类任务伴随着数据库的增长，一步一步提升到了今天的水平。

现在在 imagenet 这样的超过 1000 万图像，2 万类的数据集中，计算机的图像分类水准已经超过了人类。

图像分类，顾名思义，就是一个模式分类问题，它的目标是将不同的图像，划分到不同的类别，实现最小的分类误差。

总体来说，对于二分类的问题，图像分类可以分为**跨物种语义级图像分类，子类细粒度图像分类，以及实例级图像分类**三大类别。

跨物种分类

细粒度分类

实例级分类

传统机器学习方法：

通过各种经典的特征算子+经典分类器组合学习，比如 HoG+SVM。

深度学习方法：

各种分类网络，最为大家熟知的就是 ImageNet 竞赛了。

2012 年 Alexnet 诞生，意味着 GPU 训练时代的来临。

Alexnet 是第一个真正意义上的深度网络，与 LeNet5 的 5 层相比，它的层数增加了 3 层，网络的参数量也大大增加，输入也从 32 变成了 224。

2014 年 VGG 诞生，它共包含参数约为 550M。全部使用 3*3*的卷积核*和 2*2 的最大池化核，简化了卷积神经网络的结构。VGG 很好的展示了如何在先前网络架构的基础上通过增加网络层数和深度来提高网络的性能，网络虽然简单，但是却异常的有效，在今天 VGG 仍然被很多的任务选为基准模型。

同一年 GoogleNet 诞生，也被成为 Inception Model，它的核心是 Inception Module。一个经典的 inception 结构，包括有四个成分，1*1 卷积，3*3 卷积， 5*5 卷积，3*3 最大池化，最后对运算结果进行通道上组合，可以得到图像更好的表征。自此，深度学习模型的分类准确率已经达到了人类的水平(5%~10%)。

2015 年，ResNet 被提出。ResNet 以 3.57%的错误率表现超过了人类的识别水平，并以 152 层的网络架构创造了新的模型记录。由于 resnet 采用了跨层连接的方式，它成功的缓解了深层神经网络中的梯度消散问题，为上千层的网络训练提供了可能。

2016 年 ResNeXt 诞生，101 层的 ResNeXt 可以达到 ResNet152 的精确度，却在复杂度上只有后者的一半，核心思想为分组卷积。即首先将输入通道进行分组，经过若干并行分支的非线性变换，最后合并。

在 resnet 基础上，密集连接的 densenet 将前馈过程中将每一层与其他的层都连接起来。对于每一层网络来说，前面所有网络的特征图都被作为输入，同时其特征图也都被其他网络层作为输入所利用。

2017 年，也是 imagenet 图像分类比赛的最后一年，senet 获得了冠军。这个结构，仅仅使用了“特征重标定”的策略来对特征进行处理，也就是通过学习获取每个特征通道的重要程度，根据重要性去抑制或者提升相应的特征。

1.2 方向特点

图像分类的比赛基本落幕，也接近算法的极限。但是在实际的应用中却面临着比比赛中更加复杂，比如样本不均衡，分类界面模糊，未知类别等。如果想了解更多，请查看往期文章。

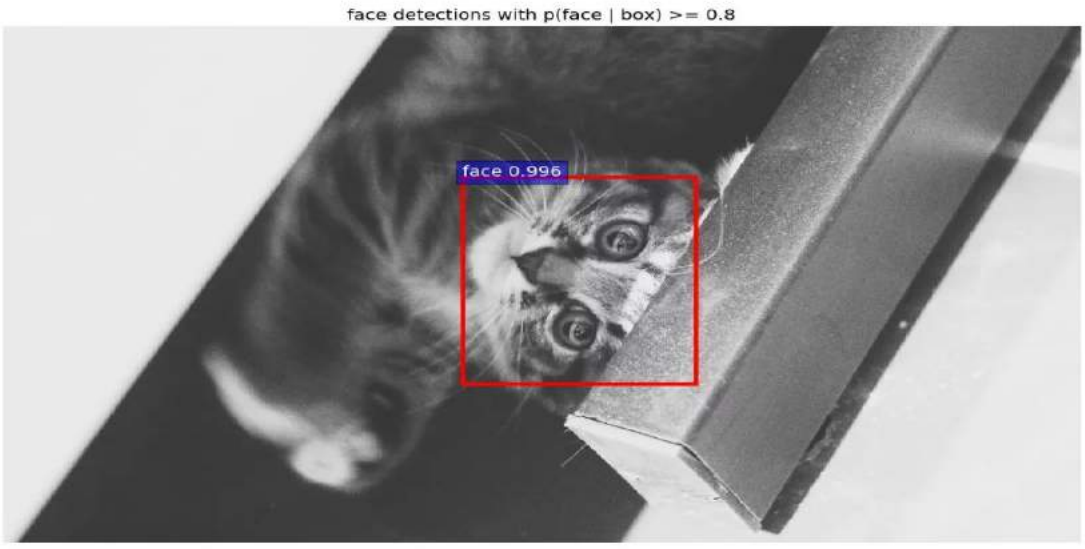
【技术综述】你真的了解图像分类吗？

2 目标检测

2.1 基本概念

分类任务给出的是整张图片的内容描述，而目标检测任务则关注图片中特定的目标。

检测任务包含两个子任务，其一是这一目标的类别信息和概率，它是一个分类任务。其二是目标的具体位置信息，这是一个定位任务。



与计算机视觉领域里大部分的算法一样，目标检测也经历了从**传统的人工设计特征和浅层分类器的思路（以）**，到大数据时代使用**深度神经网络进行特征学习的思路**。

在传统方法时代，很多的任务不是一次性解决，而是需要多个步骤的。而深度学习时代，很多的任务都是采用 End-To-End 的方案，即输入一张图，输出最终想要的结果，

算法细节和学习过程全部丢给了神经网络，这一点在物体检测这个领域，体现得尤为明显。

不管是清晰地分步骤处理，还是深度学习的 end-to-end 的方法，目标检测算法一定会有 3 个模块。**第一个是检测窗口的选择，第二个是图像特征的提取，第三个是分类器的设计。**

2.2 方法分类

传统机器学习方法：

以保罗·维奥拉和迈克尔·琼斯于2001年提出的维奥拉-琼斯目标检测框架为代表，这是第一篇基于 Haar+Adaboost 的检测方法，也是首次把检测做到实时的框架，此方法在 opencv 中被实现为 cvHaarDetectObjects()，是 opencv 中最为人熟知的目标检测方法。速度非常快，检测召回率相对如今的算法较低。

深度学习方法：

仍然要解决区域选择、提取特征、分类回归三个问题。但是在演变过程中，却发展出了 multi-stage 和 one-stage 的方法。其中 multi-stage 方法，是分步骤完成上面的任务，甚至可能需要单独训练各个网络。而 one-stage 则是一步到位。

RCNN 的框架是 multi-stage 方法的典型代表。它使用了 Selective search 先生成候选区域再检测，候选窗口的数量被控制在了 2000 个左右。选择了这些图像框之后，就可以将对应的框进行 resize 操作，然后送入 CNN 中进行训练。由于 CNN 非常强大的非线性表征能力，可以对每一个区域进行很好的特征表达，CNN 最后的输出，使用多个分类器进行分类判断。该方法将 PASCAL VOC 上的检测率从 35.1% 提升到了 53.7%，其意义与 Alexnet 在 2012 年取得分类任务的大突破是相当的，对目标检测领域影响深远。

随后 Fast R-CNN 提出 RoIPooling 从整图对应的卷积特征图选取区域特征，解决了重复提取特征的问题。Faster R-CNN 则提出 Region Proposal, anchors 把一张图片划分成 n*n 个区域，每个区域给出 9 个不同 ratio 和 scale 的 proposal，解决了重复提取候选 proposal 的问题。 RCNN 系列在工业届应用非常广泛，因此从事目标检测的同学必须掌握。

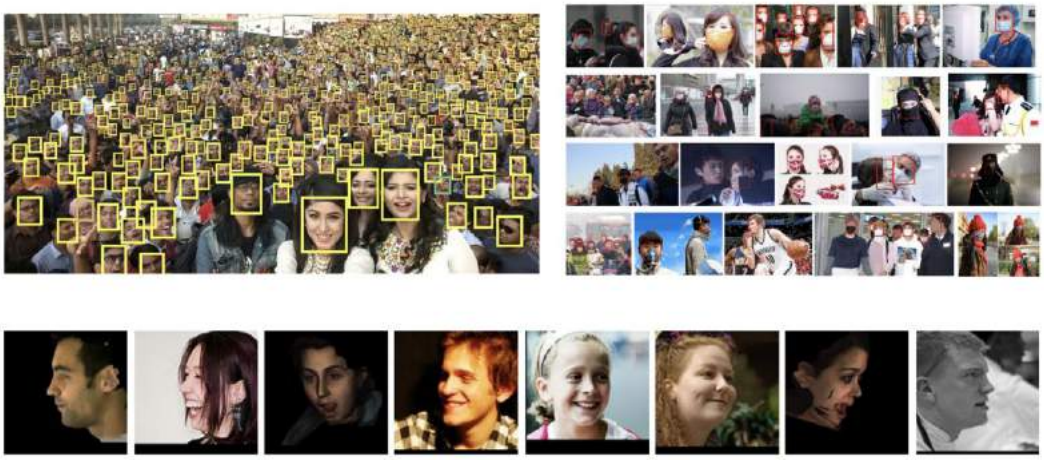
除了 multi-stage 方法，还有 one-stage 方法。以 YOLO 为代表的方法，没有显式的候选框提取过程。它首先将图片 resize 到固定尺寸，将输入图片划分成一个 7x7 的网格，每个网格预测 2 个边框，对每一个网络进行分类和定位。YOLO 方法也经过了许多版本的发展，从 YOLO v2 到 YOLO v3。YOLO 的做法是速度快，但是会有许多漏检，尤其是小的目标。所以 SSD 就在 YOLO 的基础上添加了 Faster R-CNN 的 Anchor 概念，并融合不同卷积层的特征做出预测。虽然 YOLO 和 SSD 系列的方法没有了 region proposal 的提取，速度更快，但是必定会损失信息和精度。

如果想了解更多，可以去阅读我们的往期文章。

【技术综述】一文道尽 R-CNN 系列目标检测
【技术综述】万字长文详解 Faster RCNN 源代码

2.3 方向特点

目标检测方向有一些固有的难题，比如**小脸，遮挡，大姿态**。



而在方法上，**多尺度与级联网络的设计，难样本的挖掘，多任务** loss 等都是比较大的研究小方向，咱们也写过一些文章，感兴趣的朋友可以去翻。

3 图像分割

3.1 基础概念

图像分割属于图像处理领域最高层次的图像理解范畴。所谓图像分割就是把图像分割成具有相似的颜色或纹理特性的若干子区域，并使它们对应不同的物体或物体的不同部分的技术。这些子区域，组成图像的完备子集，又相互之间不重叠。

在图像处理中，研究者往往只对图像中的某些区域感兴趣，在此基础上才有可能对目标进行更深层次的处理与分析，包括对象的数学模型表示、几何形状参数提取、统计特征提取、目标识别等。

传统方法：

图像分割问题最早来自于一些文本的分割，医学图像分割。在文本图像分割中，我们需要切割出字符，常见的问题包括指纹识别，车牌识别；由于这一类问题比较简单，因为基于阈值和聚类的方法被经常使用。

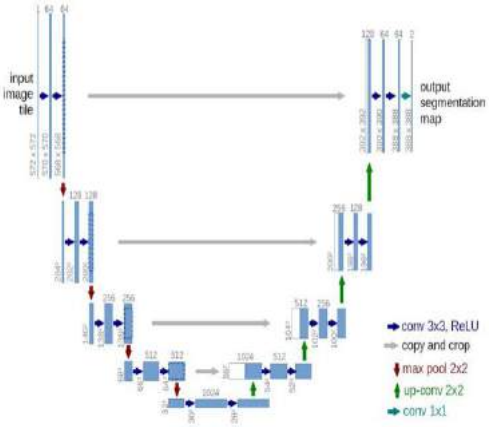
基于阈值和聚类的方法虽然简单，但因此也经常失效。以 graphcut 为代表的方法，是传统图像分割里面鲁棒性最好的方法。Graphcut 的基本思路，就是建立一张图，其中以图像像素或者超像素作为图像顶点，然后移除一些边，使得各个子图不相连从而实现分割。图割方法优化的目标是找到一个切割，使得移除边的和权重最小。

深度学习方法：

全卷积神经网络(Fully connected Network)是第一个将卷积神经网络正式用于图像分割问题的网络。

一个用于分类任务的深度神经网络通过卷积来不断抽象学习，实现分辨率的降低，最后从一个较小的 featuremap 或者最后的特征向量，这个 featuremap 通常为 5*5 或者 7*7 等大小。而图像分割任务需要恢复与原尺度大小一样的图片，所以，需要从这个 featuremap 恢复原始图片尺寸，这是一个上采样的过程。由于这个过程与反卷积是正好对应的逆操作，所以我们通常称其为反卷积。

实际上并没有反卷积这样的操作，在现在的深度学习框架中，反卷积通常有几种实现方式，一个是双线性插值为代表的插值法，一个是**转置卷积**。



3.2 方向特点

在基于深度学习的图像分割中，有一些比较关键的技术，包括**反卷积的使用，多尺度特征融合，crf 等后处理**方法。

多尺度与上下文信息：

多尺度的信息融合可以从特征图，还可以直接采用多尺度的输入图像，不过这两者本质上没有太多的差异。使用金字塔的池化方案可实现不同尺度的感受野，它能够起到将局部区域上下文信息与全局上下文信息结合的效果。对于图像分割任务，全局上下文信息通常是与整体轮廓相关的信息，而局部上下文信息则是图像的细节纹理，要想对多尺度的目标很好的完成分割，这两部分信息都是必须的。

CRF：

由于经典的 cnn 是局部的方法，即感受野是局部而不是整个图像。另一方面，cnn 具有空间变换不变性，这也降低了分割的边缘定位精度。针对 cnn 的这两个缺陷，crf 可以进行很好的弥补。crf 是一种非局部的方法，它可以融合 context 信息，Deeplab 系列就使用了 cnn 加上全连接的 crf 的方式。

另一方面，前面我们说的图像分割，是属于硬分割，即每一个像素都以绝对的概率属于某一类，最终概率最大的那一类，就是我们所要的类别。但是，这样的分割会带来一些问题，就是边缘不够细腻，当后期要进行融合时，边缘过渡不自然。此时，就需要用到 image matting 技术。

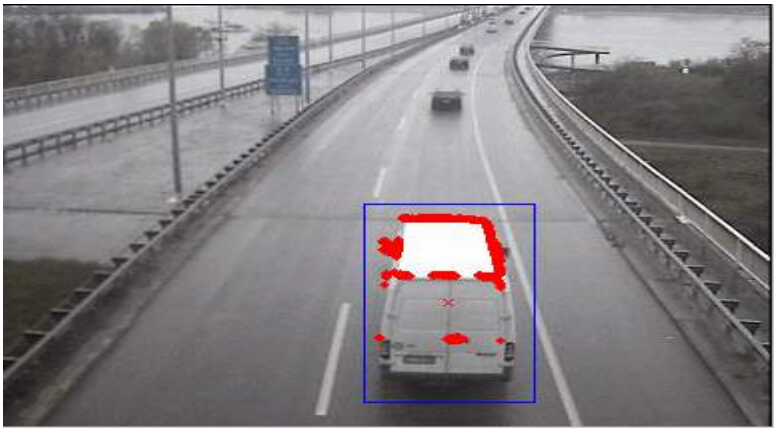
更多请查看往期文章：

【技术综述】闲聊图像分割这件事儿

4 目标跟踪

4.1 基本概念

目标跟踪，指的其实就是视频中运动目标的跟踪，跟踪的结果通常就是一个框。目标跟踪是视频监控系统中不可缺少的环节。



根据目标跟踪方法建模方式的不同，可以分为**生成式模型方法与判别式模型方法**。

生成式模型跟踪算法以均值漂移目标跟踪方法和粒子滤波目标跟踪方法为代表，判别式模型跟踪算法以相关滤波目标跟踪方法和深度学习目标跟踪方法为代表。

生成类方法：

在原始影像帧中对目标按指定的方法建立目标模型，然后在跟踪处理帧中搜索对比与目标模型相似度最高的区域作为目标区域进行跟踪。算法主要对目标本身特征进行描述，对目标特征刻画较为细致，但忽略背景信息的影响。在目标发生变化或者遮挡等情况下易导致失跟现象。

判别类方法：

通过对原始影像帧，对目标及背景信息进行区分建立判别模型，通过对后续影像帧搜索目标进行判别是目标或背景信息进而完成目标跟踪。

判别类方法与生成类方法的根本不同在于判别类方法考虑背景信息与目标信息区分来进行判别模型的建立，由于判别类方法将背景与目标进行区分，因此该类方法在目标跟踪时的表现通常更为鲁棒，目前已经成为目标跟踪的主流跟踪方式。判别类方法包括相关滤波，深度学习方法。

4.2 方向特点

目标跟踪有一些难点：

(1) 目标表征表达问题，虽然深度学习方法具有很强的目标表征能力，但是仍然容易受相似环境的干扰。

(2) 目标快速运动，由于很多跟踪的物体都是高速运动，因此既要考虑较大的搜索空间，也要在保持实时性的前提下减小计算量。

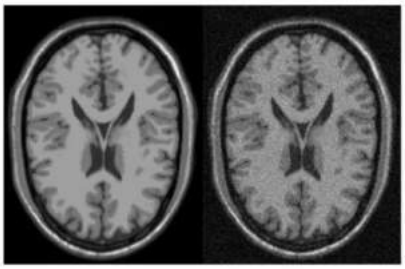
(3) 变形，多尺度以及遮挡问题，当目标发生很大的形变或者临时被遮挡如何保持跟踪并且在目标重新出现时恢复跟踪。

5 图像滤波与降噪

5.1 基本概念

现实中的数字图像在数字化和传输过程中常受到成像设备与外部环境噪声干扰等影响，称为含噪图像或噪声图像。减少数字图像中噪声的过程称为图像降噪，有时候又称为图像去噪。

降噪可以应用于图像增强和美颜等领域。



传统方法：

传统降噪算法根据降噪的原理不同可分为基于邻域像素特征的方法，基于频域变换的方法，和基于特定模型的方法。

基于空域像素特征的方法，是通过分析在一定大小的窗口内，中心像素与其他相邻像素之间在灰度空间的直接联系，来获取新的中心像素值的方法，因此往往都会存在一个典型的输入参数，即滤波半径 r。此滤波半径可能被用于在该局部窗口内计算像素的相似性，也可能是一些高斯或拉普拉斯算子的计算窗口。在邻域滤波方法里面，最具有代表性的滤波方法有以下几种：算术均值滤波与高斯滤波，统计中值滤波，双边滤波，非局部平均滤波方法，BM3D 算法。

深度学习方法：

在 2012 年，随着 Alexnet 的出现，深度学习做去噪的工作取得了一些进展，可以达到和 BM3D 差不多的水平。对于仿真的噪声和固定的噪声，深度学习已经可以很好的去除，达到或超过传统领域里最好的算法。

利用卷积神经网络去除噪声的原理很简单，输入是一张有噪声的图，标签是一张无噪声的图，输出是一张降噪后的图，损失函数是无噪声 groundtruth 与网络输出的 L2 距离，网络通常就是与图像分割算法一样的网络，卷积+与之对称的反卷积。

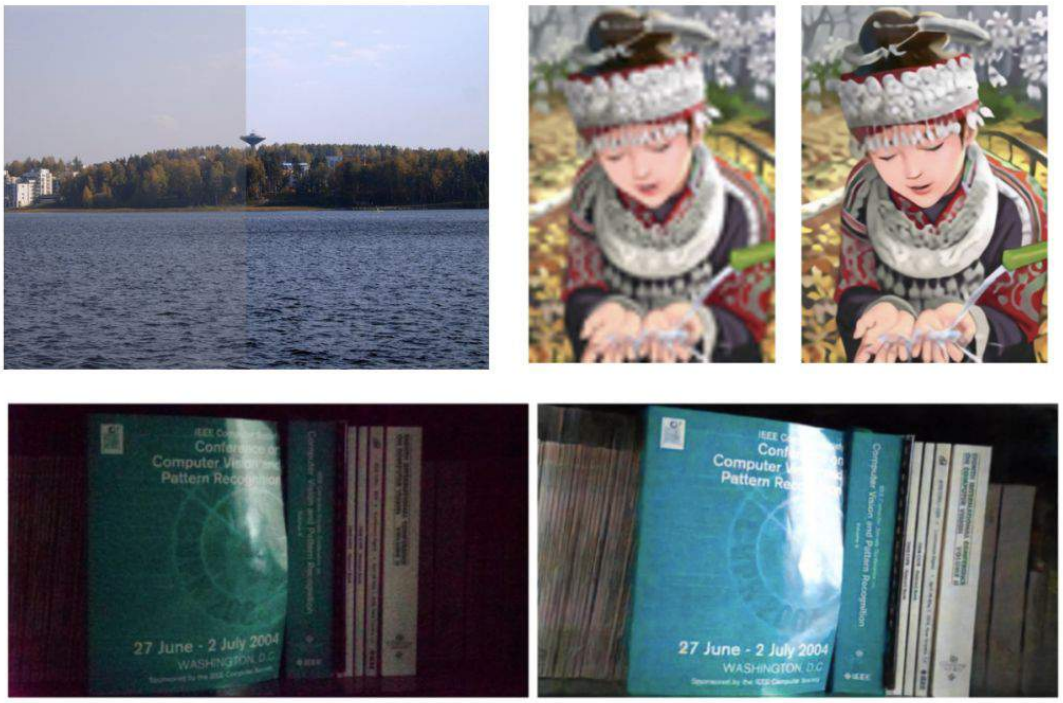
5.2 方向特点

降噪的研究聚焦在真实数据的去噪声，因为真实世界的噪声不符合高斯加性噪声的假设，而且是依赖于信息本身的。不过，真实噪声图像和相应的无噪声图像获取是非常困难，慢慢的也有了一些 benchmark，大家以后关注我们就知道了。

6 图像增强

6.1 基本概念

图像增强，即增强图像中的有用信息，改善图像的视觉效果。



图像增强实际上包含了很多的内容，上面的降噪也属于其中，只是因为降噪多了美颜这一个应用单独拿出来说一下。

对比度增强，用于扩大图像中不同物体特征之间的差别，抑制不感兴趣的特征，可用于改善图像的识别效果，满足某些特殊分析。

超分辨，使图像变得更加清晰，可以用于视频的传输先进行降采样，再进行升采样，即降低了传输成本，又增加了视觉效果。

图像修复，重建图像和视频中丢失或损坏的部分，也被称为图像插值或视频插值，主要是替换一些小区域和瑕疵，如 photoshop 中的印章工具。随着发展，已经从原先针对划痕、污点等的修复到现在对图像、视频中文字、物体等的移除，比如水印等。

传统方法：

传统的方法就是一个预定义好的非线性变换，主要有三大类方法，一类是点操作，一类是直方图操作，一类是 Retinex 理论。

点操作也被称为直接对比度增强，将每个像素独立操作，包括对数变化，指数变化，负图像，阈值化等。我们熟知的 gamma 变换如下，可以进行不同形状的映射。

直方图操作也被称为间接对比度增强，包括直方图均衡，直方图匹配等。直方图均衡化通常用来增加图像的全局对比度，尤其是当图像中主体和背景对比度相当接近的时候。直方图均衡化的效果就是让直方图更均衡的分布，这种方法对于背景和前景都太亮或者太暗的图像非常有用，通常是曝光过度或者曝光不足的图片。

Retinex 理论，即颜色恒常知觉的计算理论，Retinex 是一个合成词，它的构成是 retina（视网膜）+cortex（皮层），它将图像认为是 reflectance 和 illumination 的点乘，理论基础是在不同的照明条件下，物体的色彩不受光照非均性的影响是恒定的，而物体的颜色是由物体对长波、中波和短波光线的反射能力决定的而不是由反射光强度的绝对值决定。

深度学习方法：

以增强对比度为例，深度学习方法使用了 CNN 来进行非线性变换的学习，而且通常不仅仅局限在对比度增强，经常会同时学习到降噪。深度学习的方法有两种，一种是采用成对的图片训练，比如 pix2pix，learning in the dark，缺点是没有普适性，只能对所实验的数据集有用。一种是不需要成对图片训练，只需要好图，比如 WESPE，常配合 GAN 使用。

6.2 方向特点

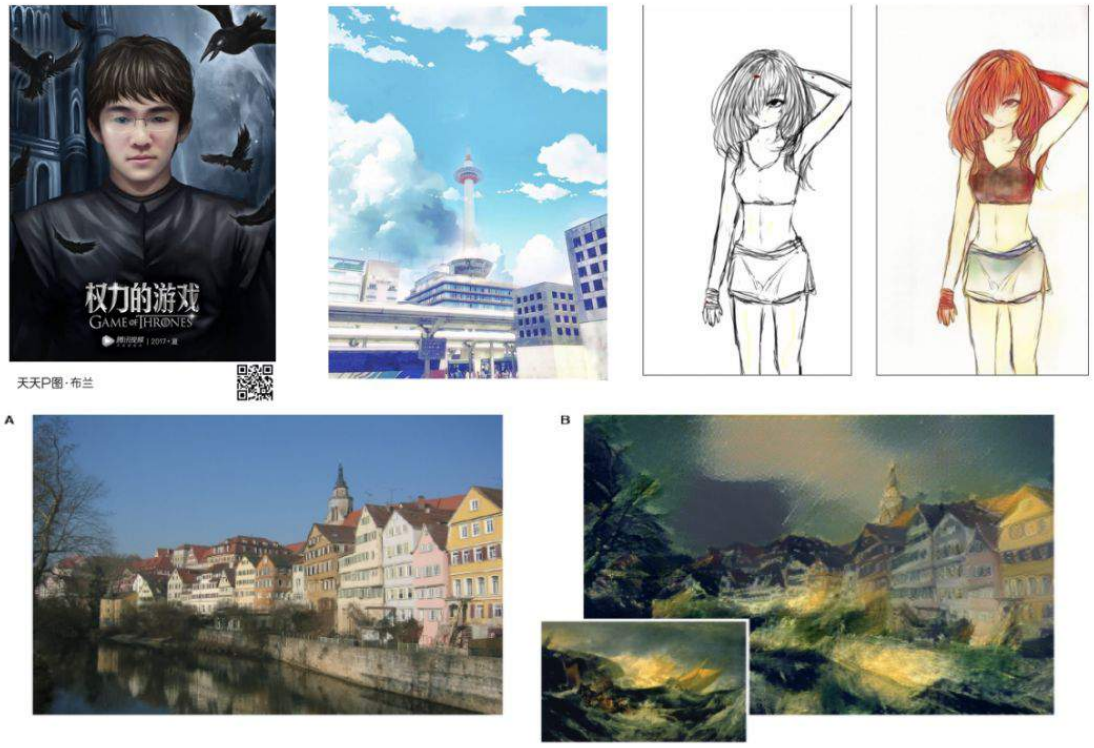
一个图像增强任务，传统方法需要分别进行降噪，颜色校正，对比度增强等各种操作，而深度学习算法的好处就是 end-to-end 输出，将整个流程丢给了网络。目前图像增强相对于前面的一些方向还是一个蓝海，覆盖的方向和应用非常广，有精力的朋友可以好好研究。

7 风格化

7.1 基本概念

图像风格化之所以引起我们的注意，完全是因为 2015 年的一个研究，可以将任意的图像转换为梵高的画作风格。 也是得益于深度学习技术的发展，传统的方法做不到这么好的效果。而随着美图秀秀，天天 P 图等 app 层出不穷的滤镜，风格化已经成为了单独的一个研究领域。

图像风格化是一个综述性的技术应用，为了简单起见，就理解为艺术类滤镜把，它指通过算法，将数码相机拍摄的照片，变成绘画、素描等艺术类的非数码相机效果，是后期程度最深的操作，将彻底改变相片的风格。



深度学习方法：

以 A Neural Algorithm of Artistic Style 论文发表为起始，Prisma 滤镜为典型代表。虽然风格迁移技术的发展日新月异，但是最革命性的还是该文章的方法，这是德国图宾根大学的研究，它通过分析某种风格的艺术图片，能将图片内容进行分离重组，形成任意风格的艺术作品，最开始的时候需要将近一个小时来处理。

就是把一幅图作为底图，从另外一幅画抽取艺术风格，重新合成新的艺术画，可以参考上面的图。

研究者认为，图片可以由**内容层（Content）与风格层（Style）两个图层描述**，相互分离开。在图像处理中经常将图像分为粗糙层与细节层，即前者描述图像的整体信息，后者描述图像的细节信息，具体可以通过高斯金字塔来得到。

卷积神经网络的各个神经元可以看做是一个图像滤波器，而输出层是由输入图像的不同滤波器的组合，深度由浅到深，内容越来越抽象。

底层信息重建，则可以得到细节，而从高层信息重建，则得到图像的”风格“。因此，可以选择两幅图像，一幅构建内容信息，一幅构建风格信息，分别进行 Content 重建与 Style 重建。通过将内容与风格组合，可以得到新的视觉信息更加有意思的图像，如计算机油画，这就是它的基本原理。方法的核心在于损失函数的设计，包括内容损失和风格损失。

内容损失在像素空间，要求风格化后的图能够保证内容的完整性。风格损失使用 vgg 特征空间的 gram 矩阵，这样就有了较高的抽象层级，实践结果表明可以很好的捕捉风格。

7.2 方向特点

如今风格化方法在很多地方都有应用，比如大家熟悉的变脸等。方法也演变成了几个方向；

（1）单模型单风格，即一个网络只能做一种风格化。

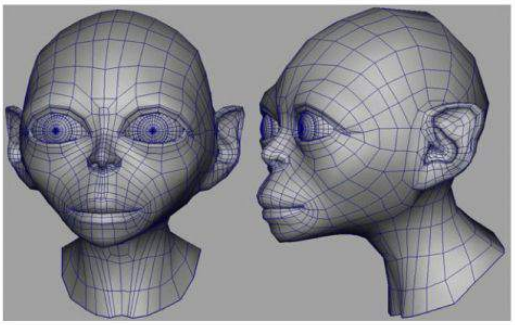
（2）单模型多风格，即一个网络可以实现多种风格，比（1）实用的多。

（3）单模型任意风格，即一个网络可以任意风格，视输入图像而定，这是最好的，更多的研究我们以后会开专题。

8 三维重建

8.1 基本概念

什么是三维重建呢？广义上来说，是建立真实世界的三维模型。随着软硬件的成熟，在电影，游戏，安防，地图等领域，三维重建技术的应用越来越多。目前获取三维模型的方法主要包括三种，**手工建模，仪器采集与基于图像的建模**。



(1) 手工建模作为最早的三维建模手段，现在仍然是最广泛地在电影，动漫行业中应用。顶顶大名的 3DMax 就是典型代表，当然了，它需要专业人士来完成。

(2) 由于手工建模耗费大量的人力，三维成像仪器也得到了长期的研究和发展。基于结构光（structured light）和激光扫描技术的三维成像仪是其中的典型代表。这些基于仪器采集的三维模型，精度可达毫米级，是物体的真实三维数据，也正好用来为基于图像的建模方法提供评价数据库。由于仪器的成本太高，一般的用户是用不上了。

(3) 基于图像的建模技术（image based modeling），顾名思义，是指通过若干幅二维图像，来恢复图像或场景的三维结构，这些年得到了广泛的研究。

我们这里说的三维重建，就特指**基于图像的三维重建方法**，而且为了缩小范围，只说人脸图像，并简单介绍其中核心的 3DMM 模型。

3DMM 模型：

人脸三维重建方法非常多，有基于一个通用的人脸模型，然后在此基础上进行变形优化，会牵涉到一些模板匹配，插值等技术。有基于立体匹配（各种基于双目，多目立体视觉匹配）的方法，通过照相机模型与配准多幅图像，坐标系转换，获取真实的三维

坐标，然后进行渲染。有采用一系列的人脸作为基，将人脸用这些基进行线性组合的方法，即 Morphable models 方法。

其中，能够融会贯通不同传统方法和深度学习方法的，就是 **3D Morphable Models 系列**方法，从传统方法研究到深度学习。

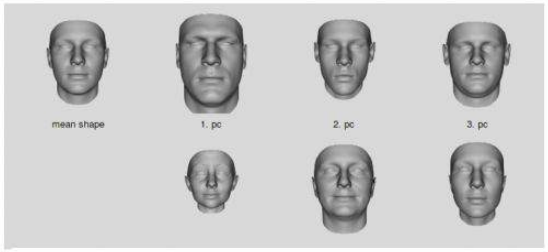
它的思想就是一幅人脸可以由其他许多幅人脸加权相加而来，学过线性代数的就很容易理解这个正交基的概念。我们所处的三维空间，每一点(x,y,z)，实际上都是由三维空间三个方向的基量，(1,0,0)，(0,1,0)，(0,0,1)加权相加所得，只是权重分别为 x,y,z。

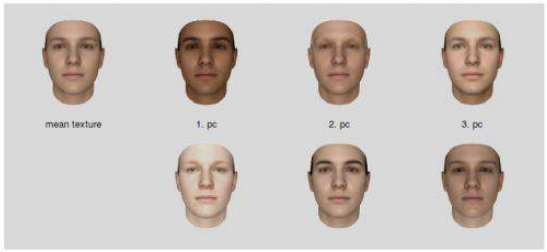
转换到三维空间，道理也一样。每一个三维的人脸，可以由一个数据库中的**所有人脸组成的基向量空间中进行表示**，而求解任意三维人脸的模型，实际上等价于求解各个基向量的系数的问题。

每一张人脸可以表示为：

形状向量 Shape Vector：S=(X1,Y1,Z1,X2,Y2,Z2,...,Yn,Zn)

纹理向量 Texture Vector：T=(R1,G1,B1,R2,G2,B2,...,Rn,Bn)





而一张任意的人脸，其等价的描述如下：

$$S_{model} = \overline{S} + \sum_{i=1}^{m-1} \alpha_i s_i, \quad T_{model} = \overline{T} + \sum_{i=1}^{m-1} \beta_i t_i$$

其中第一项 Si，Ti 是形状和纹理的平均值，而 si，ti 则都是 Si，Ti 减去各自平均值后的协方差矩阵的特征向量。 基于 3DMM 的方法，都是在求解 α，β 这一些系数，当然现在还会有表情，光照等系数，但是原理都是通用的。

原理就说到这里，我们以后会专门讲述。

8.2 方向特点

人脸的三维建模有一些独特的特点。

（1）预处理技术非常多，人脸检测与特征点定位，人脸配准等都是现在研究已经比较成熟的方法。利用现有的人脸识别与分割技术，可以缩小三维人脸重建过程中需要处理的图像区域，而在有了可靠的关键点位置信息的前提下，可以建立稀疏的匹配，大大提升模型处理的速度。

（2）人脸共性多。正常人脸都是一个鼻子两只眼睛一个嘴巴两只耳朵，从上到下从左到右顺序都不变，所以可以首先建立人脸的参数化模型，实际上这也是很多方法所采用的思路。

人脸三维重建也有一些困难。

（1）人脸生理结构和几何形状非常复杂，没有简单的数学曲面模型来拟合。

（2）光照变化大。同一张脸放到不同的光照条件下，获取的图像灰度值可能大不一样的，这些都会影响深度信息的重建。

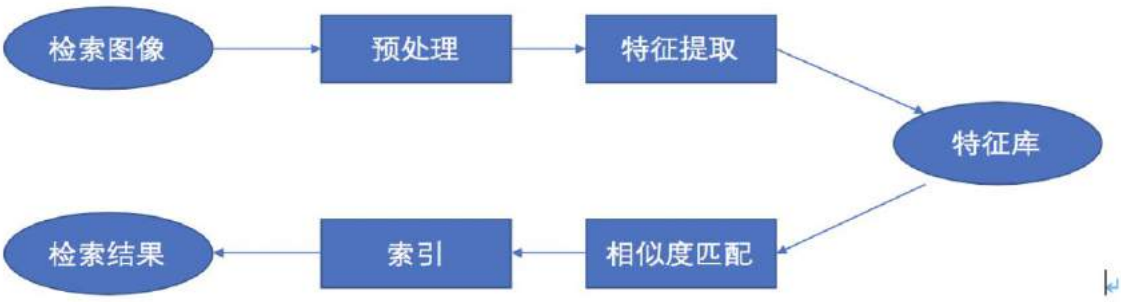
（3）特征点和纹理不明显。图像处理最需要的就是明显的特征，而光滑的人脸除了特征关键点，很难在脸部提取稠密的有代表性的角点特征。这个特点，使得那些采用人脸配准然后求取三维坐标的方法面临着巨大的困难。

9 图像检索

9.1 基本概念

图像检索的研究从 20 世纪 70 年代就已经开始，在早期是基于文本的图像检索技术（Text-based Image Retrieval，简称 TBIR），利用文本来描述图像的特征，如绘画作品的作者、年代、流派、尺寸等。随着计算机视觉技术的发展，90 年代开始出现了对图像的内容语义，如图像的颜色、纹理、布局等进行分析和检索的图像检索技术，也就是**基于内容的图像检索**（Content-based Image Retrieval，简称 CBIR）技术，本小节的图像检索就特指基于内容的图像检索。

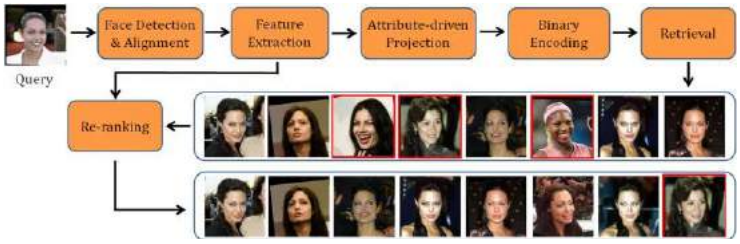
基于内容的图像检索也经历了传统方法和深度学习方法两个主要阶段，传统的基于内容的图像检索通常包括以下流程：



预处理，通常包括一些图像归一化，图像增强等操作。特征提取，即提取一些非常鲁棒的图像特征，比如 SIFT，HoG 等特征。特征库就是要查询的库，库中不存储图像而

是存储特征，每一次检索图像完成特征提取之后，就在特征库中进行匹配和相似度计算。索引就是在某种相似性度量准则下计算查询向量到特征库中各个特征的相似性大小，最后按相似性大小进行高效的排序并顺序输出对应的图片。

图像检索的中最复杂的一步就是检索，在这一步完成验证过程。



最简单的方法就是暴力(brute-force) 搜索方法(又称线性扫描)，即逐个与数据库中的每个点进行相似性计算然后进行排序，这种简单粗暴的方式虽然很容易实现，但是会随着数据库的大小以及特征维度的增加其搜索代价也会逐步的增加，从而限制在数据量小的小规模图像数据库，在大规模图像库上这种暴力搜索的方式不仅消耗巨大的计算资源，而且单次查询的响应时间会随着数据样本的增加以及特征维度的增加而增加，为了降低搜索的空间的空间复杂度与时间复杂度，研究者们提出了很多高效的检索技术，其中最成功的大家也最熟悉到方法**是基于哈希的图像检索方法**。

深度学习在图像检索里面的作用就是**把表征样本的特征学习好**，就够了。

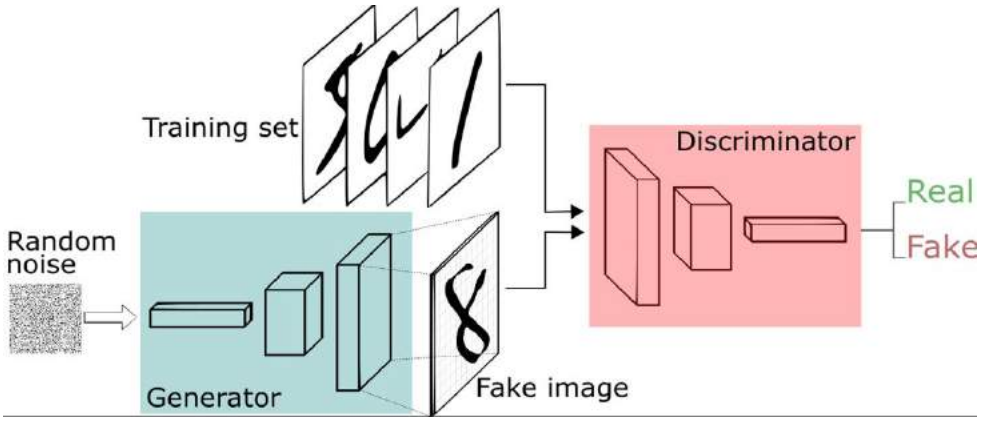
9.2 方向特点

图像检索系统具有非常大的商业价值，从搜索引擎的以图搜图，到人脸验证和识别系统，到一些搜索排序系统(比如基于美学的摄影图库)。由于图像特征的学习是一个通用的研究方向，因此更多的在于设计高效的检索系统。

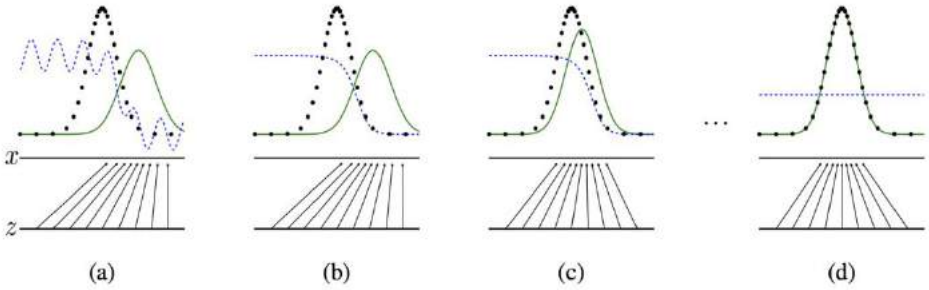
10 GAN

10.1 基本概念

GAN，即 Generative adversarial net，被誉为新的深度学习，涉及的研究非常多，可以单列为一个方向，一个经典的网络结构如下。



GAN 的原理很简单，它包括两个网络，一个生成网络，不断生成数据分布。一个判别网络，判断生成的数据是否为真实数据。



上图是原理展示，黑色虚线是真实分布，绿色实线是生成模型的学习过程，蓝色虚线是判别模型的学习过程，两者相互对抗，共同学习到最优状态。

关于 GAN 的基础，我们以前已经写过相关的内容，大家去看就可以了。

【技术综述】有三说 GANs（上）

10.2 方向特点

作为新兴和热门方向，GAN 包含的研究方向非常的广，包括 GAN 的应用，GAN 的优化目标，GAN 的模型发展，GAN 的训练技巧，GAN 的理论分析，GAN 的可视化等等，以后等着我们的分享即可。

11 总结

深度学习彻底点燃和推进了计算机视觉各大领域的研究，这是个可以投以终身的行业，希望你会喜欢，别忘了持续关注我们噢。

【AI 白身境】AI+，都加在哪些应用领域了

本篇是**《AI 白身境》**的第十一篇，所谓白身，就是什么都不会，还没有进入角色。要说现在什么最火，那都不用说，肯定是 AI。AI 已经渗入到了我们生活的方方面面。除了大家熟知的自动驾驶汽车、图像美颜，聊天机器人等，还有许多方面都应用到了 AI，本篇就和大家聊下 AI 当前在各大领域的应用。

作者|汤兴旺 言有三

1 AI+交通

既然要说 AI+交通，那就不得不提自动驾驶了，自动驾驶作为 AI 与制造业的一大产物，目前各大主机厂和 IT 公司都想在自动驾驶领域分一杯羹。另外值得注意的是习近平总书记在 2019 年 1 月 21 日的中央党校的一个研讨班的开班仪式上强调要加快“自动驾驶的立法”，这说明什么，应该不需要我多言。自动驾驶在未来绝对是最热门的领域之一，当然自动驾驶发展的怎么样，必须要看 AI 能发展到什么程度。

上图是 2018 年央视春晚珠海分会场，由百度 Apollo 无人车领衔的百余辆无人车精彩亮相的画面。是不是很震撼。

当前自动驾驶技术的关键还是在环境感知、决策规划和控制这三个方面。而要想环境感知有更好的发展，以深度学习为代表的 AI 技术如计算机视觉等非常重要。

2 AI+医疗

根据美国著名市场调研机构 Tractica 的数据显示，预计到 2025 年，医疗机构将为人工智能支付超过 340 亿美元，2018 年为 21 亿美元。这说明医疗卫生行业对 AI 的迫切需要！

医疗卫生行业对 AI 的迫切需求主要在以下几个方面：

1、医学数据分析需求
AI 能通过患者数据训练的深度学习诊疗模型，快速筛选出适合患者的药物和治疗方案，追踪和预测个体疾病的进程。

2、远程医疗
人工智能的影响不仅在医院内部的临床诊疗工作，还将提升院外服务的可及性。一旦患者离开医院，医生可能难以确保患者遵守规定的治疗计划或监测慢性健康状况，AI 能够弥合时间和空间距离，实现信息的高效传输和共享，这将帮助医生及时了解患者的健康状况。

3、促进医疗行为规范化
引入医疗 AI 工具，在诊疗流程中能督促医生按照临床诊疗规范执行医疗方法，这将大大减少不规范行为给患者带来的安全威胁。

4、新药开发
新药研发的目标是找到可调控机体生物学功能的实体物质，如小分子、大分子或生物活体等；然而优质靶点的寻找很难，而且一旦出现一个获得临床验证的新靶点，叠罗汉式的实验前仆后继并不鲜见，研发成本会是疯狂增加。为了解决这个问题，现在我们可以将人工智能的相关技术应用于疾病的靶点预测、高通量数据的分析以及系统生物学的建模过程中。

3 AI+金融

当前人工智能的应用已广泛渗透到金融行业中，且日渐成熟，并推动银行、保险、资本市场三大金融行业的深刻变革。

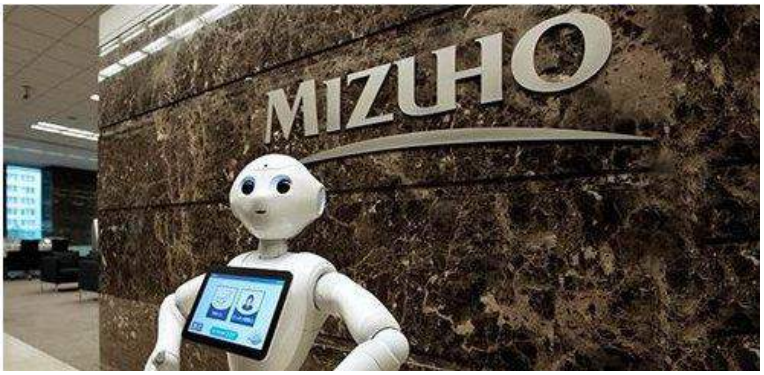
AI 在金融方面的应用主要集中在以下几个方面：

3.1、信用评分

以往的多数信用评分模型，大多数是使用金融机构的交易与支付数据，并运用回归、决策树、统计分析等工具来生成信用评分。现在，银行等金融机构日益使用新型的非结构化与半结构化的数据来源(如社交媒体、手机和短信)来捕捉借款人的信用，并用人工智能来评估消费者行为和支付意愿等定性因素，使筛选借款人的速度更快、成本更低。

3.2.金融 AI 机器人

金融 AI 机器人是用于帮助客户处理问题的虚拟助手，其使用 NLP 自然语言与客户交互，并使用机器学习算法来优化。当前许多金融服务公司在其移动应用程序或公司里面都引入了 AI 机器人。下图就是日本瑞穗银行的机器人顾问。



当前在金融业 AI 应用还有很多，如优化金融资本、模型风险管理与压力测试、投资组合管理等方面。可以说 AI 促进金融业更加智能化，更好的为公众服务。

4 AI+教育

在人工智能风潮的引领下，AI+教育的浪潮如火如荼，国内的各大教育机构如新东方、英语流利说都迅速在 AI 领域跑马圈地，试图用人工智能变革传统的教育体系。

在 AI+教育做的最多的应该就是英语流利说了，他们有个口号就是“用 AI 替代真人老师”。下图是英语流利说的英语水平测试：

据介绍其可以精准定位用户英语水平，然后量身定制系统课程。有兴趣的可以测一下水平，这个是免费的。

在 AI+教育领域还有个比较好的应用——魔镜系统。

魔镜系统是好未来自主研发的人工智能教学辅助系统。该系统可以借助摄像头捕捉学生上课时的举手、练习、听课、发言等课堂状态和面部情绪变化数据，生成专属于每一个学生的学习报告。其提供的多维度量化数据，能够真实反映出学生学习过程中的每个阶段、每种状态，帮助老师基于科学数据而调整、优化教学方案，帮助学生更好地提升专注度，实现更高效的学习，同时也可以使每一个孩子都得到充分的关注；另外它能利用表情动作识别，可以产出孩子整堂课专注度曲线以及学习状态小视频，父母即使不去陪读也可以轻松了解孩子在课堂的学习状态。

5 AI+农业

要提到农业与 AI，大家可能有点不可思议，AI 这么高科技的东西能应用到农业吗？哈哈，当然能，而且 AI 与农业的结合可以说效果显著。现在靠天吃饭的农业正在发生巨大转变，人工智能可以使农场品更安全、更营养、更值钱。

在 2018 年 6 月 7 日的云栖大会·上海峰会上，阿里云正式发布 ET 农业大脑。其目标是希望将人工智能与农业深入结合，目前已应用于生猪养殖、苹果及甜瓜种植，已具备数字档案生成、全生命周期管理、智能农事分析、全链路溯源等功能。

6 AI+新零售

大家应该都听过新零售这个词，按照阿里的解释，新零售就是以消费者体验为中心的数据驱动的泛零售形态。说简单点实际上就是 AI 与我们平时见到的零售相结合。

说到新零售就不得不提阿里旗下的盒马鲜生了，其是阿里巴巴对线下超市完全重构的新零售业态。

阿里巴巴能够为盒马鲜生的消费者提供会员服务，用户可以使用淘宝或支付宝账户注册，以便消费者从最近的商店查看和购买商品。盒马未来可以跟踪消费者购买行为，借助大数据做出个性化的建议。

除了盒马鲜生，阿里的淘咖啡也是一个新零售的典型。

用户首次进店后需通过手机淘宝扫码获得电子入场码，通过认证闸机后可以开始购物。最后离开时，用户必须经过两道门，第一道门感应到用户的离店需求时自动开启，经过几秒结算时间后开启第二道门，顾客可离店。

淘咖啡在技术上主要采用生物特征自主感知和学习系统、结算意图识别和交易系统、目标检测与追踪系统来追踪消费者在店内的行为及运动轨迹。

7 AI+家居

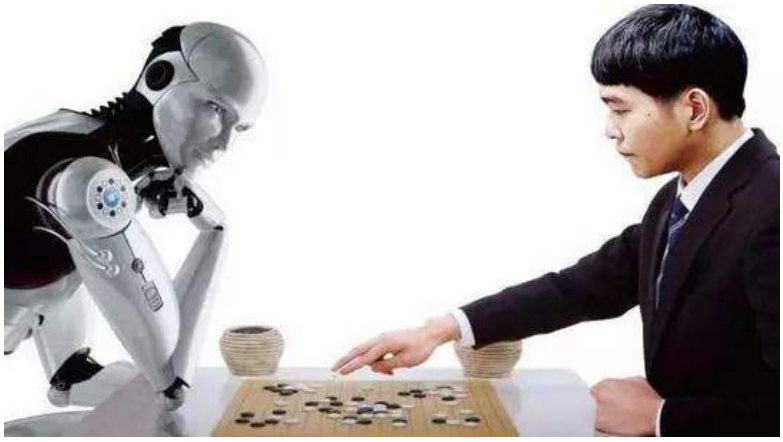
智能家居可以说是近年来最热门的话题之一。随着科技的发展，从感应采光调节室内光线的智能灯到输入指令就可自动扫地的机器人，从记录食物新鲜程度的冰箱到能识别指纹和人脸的门锁，曾经只能在科幻大片中见到的场景正渐渐成为现实。

上图是电影《钢铁侠》中无所不能的智能管家 Jarvis，而扎克伯格受此启发设计出了类似的 AI 系统来控制智能家居，也命令为 Jarvis，在 Jarvis 的帮助下，扎克伯克可以用手机和电脑来调节空调温度、室内灯光明暗，还能烤面包，从网上搜索歌曲自动播

放。另外 Jarvis 系统识别访客的功能也非常好，扎克伯格在自己家门口安装了多个摄像头，从不同角度拍摄家门口的画面。当访客靠近时，Jarvis 系统会马上识别到门口有人，然后激活程序，对访客的面部细节进行探测，接着会在 Facebook 人脸数据库中找到对应的目标，并根据扎克伯格的日程表及访客名单来判断对方是否为不速之客，确认后才会打开门，并告知扎克伯格客人已经到了。哈哈你是不是也想要有一个这样的智能家居！

8 AI+体育

在体育领域，人们开始对 AI 格外关注应该是在 2016 年李世石和谷歌围棋 AI “AlphaGo”的比赛，根据媒体报道，比赛的第一天吸引了全球共计 1 亿人次观看，这应该是 AI 在体育领域第一次大规模的爆发。



现在体育行业有许多方面都涉及了 AI 技术。下面和大家介绍一些体育方面的 AI，如下：

8.1、比赛精彩视频引入 AI 技术

在 2018 年俄罗斯世界杯开赛前夕，IBM 和美国福克斯体育（Fox Sports）合作制作一个 AI 平台。IBM 的解决方案包括分析视觉、音频、文本数据，福克斯体育将利用 IBM 沃森媒体的功能来进行自动高级元数据标记以及快速筛选新的和存档视频片段，以识别比赛时刻并生成实时的精彩视频。

现在在腾讯体育的NBA直播比赛中同样也有这样的AI技术，他们利用IBM AI Vision视觉大脑来快速完成对精彩瞬间的剪辑。

8.2、AI 裁判

国际体操联合会当前和日本通信技术公司富士通（Fujitsu）合作，计划把 AI 技术引进东京奥运会的打分系统，利用 3D 传感器接收的数据并结合 AI 分析鞍马和自由体操等体操项目，让 AI 分担一部分裁判工作。AI 技术的引入，在一定程度上，能够使评分系统更加公正。

8.3、智能运动鞋

在 2019 年的 1 月 16 日，耐克发布了新款运动鞋 Nike Adapt BB。这双篮球鞋可不一般，是一双智能篮球鞋，它可以自系鞋带，还能与智能手机应用程序配合使用以调整合脚度。

AI 在体育方面的应用还有很多，相信在未来，大数据和人工智能将是推动体育产业发展的最重要的角色之一。

9 AI+电竞

最近几年随着英雄联盟、王者荣耀、绝地求生等游戏大火，电子竞技也变的火爆起来。现在似乎每个小孩都会玩王者荣耀。哈哈，你曾经被“小学生”坑过不！这就是现在电子游戏的火爆程度。现在随着 AI 技术的发展，许多游戏也都引入了 AI，下面将介绍下 AI 在游戏中的应用。

9.1、王者荣耀 AI“绝悟”

“绝悟”的首次露面是在 2018KPL 秋季赛的总决赛上（12 月 22 日），当时“绝悟”与前 KPL 职业选手和职业解说组成的人类战队（平均水平超过 99%玩家）进行 5V5 的水平测试，并取得胜利。

这次“绝悟”的出现，不仅仅只是 KPL 秋季赛总决赛的惊鸿一瞥，更标志着我们对“AI+电竞”的全新探索。

9.2、星际争霸 2AI“AlphaStar”

AlphaStar 是谷歌的子公司 Deepmind 在游戏领域推出的一款对战型游戏 AI 作品。在 2019 年 1 月 24 日，AlphaStar 与《星际争霸 2》的两位职业选手 TLO 和 MaNa 对战，都取得了 5:0 的骄人战绩。据了解，目前的 AlphaStar 已经十分成熟了，它相当于拥有 200 年《星际争霸 2》游戏经验的玩家。

10 AI+直播

你曾想过 AI 与直播这两个新的东西碰到一起会产生怎样的火花吗？下面请看 AI 在直播中的应用。

10.1、AI+直播内容审核

直播平台利用深度学习算法，通过模拟人脑神经网络，构建具有高层次表现力的模型，能够对高复杂度数据形成良好的解读。这样在审核违法内容时能有效节省人工复审的工作，大大的提高了工作效率。

10.2、AI+直播个性化推荐

现在许多直播平台把 AI 技术实际运用到了直播之中，利用其进行内容分析，并作出智能优化，为观众提供更加优质的内容。

就比如拿目前最受欢迎的游戏之一绝地求生来说，虎牙直播运用 AI 技术建立了全新的观看模式，自动分析直播内容，让玩家更加直观的找到想看的内容。比如：决赛圈、单排、双排等功能。

11 总结

AI 与产业结合实际上才刚刚开始，每个科技公司都在蠢蠢欲动，对于我们来说，最重要的或许就是在扎实的基础上不断创新，不断跟上潮流！

【AI 白身境】究竟谁是 paper 之王，全球前 10 的计算机科学家

本篇是《AI 白身境》的第十二篇，也是最后一篇了，作为最后一篇，我的想法是激励大家见贤思齐。本来想写篇必须关注的大佬，但是实在是太难写了，人太多也容易引起争议，那就用最权威的资料来，学术界公认的 h-index 排名。

所谓 H-index，就是 high citations，简单来说就是论文被引用的频次。

作者|言有三

1 H-index 排名前十的计算机科学家

下图是 2018 年计算机科学领域的 H-index 排名前十，相信从中就是小白们也能看到不少熟悉的名字。

Author	# citations/day (2018)	# citations/day (2017)	# citations/day (2016)	H-index
Yoshua Bengio	131	101	64	121
Geoffrey Hinton	127	108	76	139
Yann LeCun	62	52	35	103
Andrew Zisserman	59	53	42	143
David Haussler	48	49	48	150
Trevor Darrell	43	39	27	106
Stephen Boyd	41	42	40	108
Michael I. Jordan	39	38	37	148
Christopher D Manning	36	36	33	106
Herbert A Simon	36	40	43	174

完整名单见 http://www.iro.umontreal.ca/~bengioy/citation-rate-CS-1sept2018.html?utm_source=wechat_session&utm_medium=social&utm_oi=635845013939032064

H-index 排名越高说明论文被人引用的越频繁，在学术界来说这就意味着影响力。下面我们来了解一下排名前十的大佬们都是谁，做过什么。

1、Yoshua Bengio，加拿大计算机科学家，深度学习三巨头之一，LeNet5 作者之一，花书《Deep learning》作者之一，一直呆在学术界。

代表性文章：

[1] Lécun, Yann, et al. “Gradient-Based Learning Applied to Document Recognition.” Proceedings of the IEEE, vol. 86, no. 11, 1998, pp. 2278－2324.

[2] Bengio Y, Courville A C, Vincent P, et al. Representation Learning: A Review and New Perspectives[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2013, 35(8): 1798-1828.

2、Geoffrey Hinton，加拿大认知心理学家和计算机科学家，深度学习三巨头之一，反向传播算法提出者之一，2006 年在 science 期刊发表深层网络逐层初始化训练方法，揭开深度学习世纪新序幕，其弟子 Alex Krizhevsky 提出 AlexNet 网络。

代表性文章：

[1] Rumelhart D E , Hinton G E , Williams R J . Learning internal representations by error propagation[M]// Neurocomputing: foundations of research. MIT Press, 1988.
[2] Hinton G E, Salakhutdinov R. Reducing the dimensionality of data with neural networks.[J]. Science, 2006, 313(5786): 504-507.
[3] Krizhevsky A , Sutskever I , Hinton G . ImageNet Classification with Deep Convolutional Neural Networks[C]// NIPS. Curran Associates Inc. 2012.

3，Yann LeCun，法国计算机科学家，深度学习三巨头之一，Facebook 首席人工智能科学家，LeNet5 网络第一作者，深度学习综述《Deep learning》作者之一。

至此三巨头都出现了，不愧是三巨头，它们之间也有着千丝万缕的合作，从上面同时出现在 LeNet5 和深度学习花书的 Yoshua Bengio 和 Yann LeCun 就可以看出，两人年纪也相当，而 Hinton 其实已经是两者的老师级别。

代表性文章：

[1] LeCun Y, Boser B, Denker J S, et al. Backpropagation applied to handwritten zip code recognition[J]. Neural computation, 1989, 1(4): 541-551.
[2] LeCun Y, Denker J S, Solla S A. Optimal brain damage[C]//Advances in neural information processing systems. 1990: 598-605.
[3] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

4、Andrew Zisserman，英国计算机科学家，牛津大学教授，计算机视觉研究员，经典书《Multiple View Geometryin Computer Vision》作者，VGG 网络作者之一，Pascal Visual Object Classes (VOC) Challenge 发起者之一，Deep Mind 研究员。

代表性文章：

[1] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. international conference on learning representations, 2015.

[2] Everingham M, Van Gool L, Williams C K, et al. The Pascal Visual Object Classes (VOC) Challenge[J]. International Journal of Computer Vision, 2010, 88(2): 303-338.

[3] Jaderberg M, Simonyan K, Zisserman A, et al. Spatial transformer networks[J]. neural information processing systems, 2015: 2017-2025.

5，David Haussler，美国生物信息学家，霍华德休斯医学研究所研究员、生物分子工程教授等，人类基因组计划竞赛中组装了第一个人类基因组序列。

代表性文章：

[1] Lander E S, Linton L, Birren B, et al. Initial sequencing and analysis of the human genome.[J]. Nature, 2001, 409(6822): 860-921.

6，Trevor Darrell，加州大学伯克利分教授，伯克利人工智能研究（BAIR）实验室的联合主任，Caffe，RCNN 作者之一，

代表性文章：

[1] Jia Y, Shelhamer E, Donahue J, et al. Caffe: Convolutional Architecture for Fast Feature Embedding[J]. acm multimedia, 2014: 675-678.
[2] Girshick R B, Donahue J, Darrell T, et al. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation[J]. computer vision and pattern recognition, 2014: 580-587.

7，StephenP.Boyd，三星工程教授，斯坦福大学信息系统实验室电气工程教授，凸优化书籍《Convex optimization》作者。

代表性文章：

[1] Stephen Boyd L V, Stephen Boyd L V. Convex optimization[J]. IEEE Transactions on Automatic Control, 2006, 51(11):1859-1859.
[2] Candes E J, Wakin M B, Boyd S P. Enhancing Sparsity by Reweighted l(1) Minimization[J]. Journal of Fourier Analysis & Applications, 2007, 14(5):877-905.

8，Michael I. Jordan，美国科学家、加州大学伯克利分校教授。机器学习领域的领军人物之一，2016 年《科学》杂志评定的世界上最具影响力的计算机科学家。Latent Dirichlet Allocation 模型作者。

代表性文章：

[1] Blei D M, Ng A Y, Jordan M I. Latent dirichlet allocation[J]. Journal of Machine Learning Research, 2012, 3:993-1022.

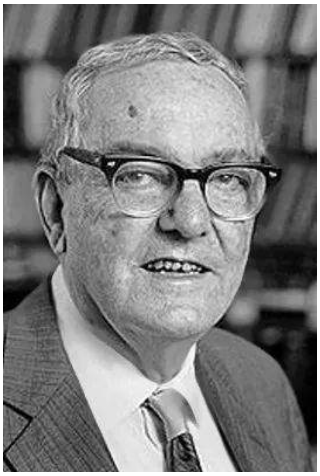
9，Christopher Manning，斯坦福大学人工智能实验室主任，语言学和计算机科学家。书籍《Introduction to information retrieval》，《Foundations of Statistical Natural Language Processing》作者。

代表性文章：

[1] Manning C D. Foundations of statistical natural language processing[M]// Foundations of Statistical Natural Language Processing. 1999.

[2] Larson R R. Introduction to Information Retrieval[J]. Journal of the Association for Information Science and Technology, 2010, 61(4): 852-853.

10，Herbert A Simon，诺贝尔经济学奖，图灵奖等获得者，书籍《The Sciences of the Artificial》，《Human Problem Solving》作者，也是唯一一个已经不在世近二十年的科学家，却还能在过去一年的论文引用前十中占据一席，可见影响力之大。



代表论文：

Publications

TITLE	CITATIONS	YEAR
The Sciences of the Artificial MIT Press	23,543	1981
Human problem solving: The state of the theory in ... American Psychologist, volume 26, issue 2, pp 14...	19,166	1971
Verbal Reports as Data. Psychological Review, volume 87, issue 3, pp 215 ...	16,447	1980
Protocol analysis : verbal reports as data MIT Press	15,656	1984
A Behavioral Model of Rational Choice Quarterly Journal of Economics, volume 69, issue ...	14,808	1955

除了上面的 10 位，计算机科学领域还有很多世界级的研究人员值得我们去关注的，比如花书作者之一和生成对抗网络的提出者 Ian Goodfellow 等，不再过多介绍。

2 深度学习领域的优秀青年华人

如果说世界级科学家离我们太遥远，那么身边优秀的华人是不是需要好好关注？下面介绍几个优秀的 80 后青年华人，都是非常有代表性的人物，对深度学习有突破性的学术贡献或开源框架作者。

1，何恺明，本科就读于清华大学，博士毕业于香港中文大学多媒体实验室，曾在微软亚洲研究院担任实习生，目前在 Facebook 人工智能实验室（FAIR）担任研究科学家。他是 Resnet、Mask R-CNN 第一作者，也是首位获计算机视觉领域三大国际会议之一 CVPR“最佳论文奖”的中国学者。另外他也获得了 CVPR 2016 和 ICCV 2017（Marr Prize）的最佳论文奖，并获得了 ICCV 2017 最佳学生论文奖，CVPR 2018 的 PAMI 年轻学者奖，这就是别人隔壁家的小明和学霸。

代表性文章：

[1] He K , Zhang X , Ren S , et al. Deep Residual Learning for Image Recognition[J]. 2015.

[2] He K, Gkioxari G, Dollar P, et al. Mask R-CNN[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2017, PP(99):1-1.

2，贾扬清，深度学习框架 Caffe 之父。本科和硕士研究生就读于清华大学，博士毕业于加州大学伯克利分校，曾在新加坡国立大学、微软亚洲研究院、NEC 美国实验室、Google Brain 工作，现任 Facebook 研究科学家，负责前沿 AI 平台的开发以及前沿的深度学习研究。

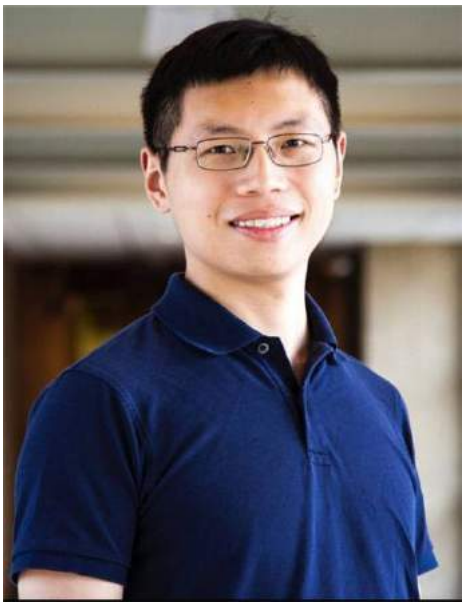
代表性文章：

[1] Jia Y , Shelhamer E , Donahue J , et al. Caffe: Convolutional Architecture for Fast Feature Embedding[J]. 2014.

[2] Decaf: A deep convolutional activation feature for generic visual recognition

如果说何凯明是学术界的青年扛把子，那么贾扬清就是工业界的青年扛把子了，他还有知乎账号，冒过几个泡。

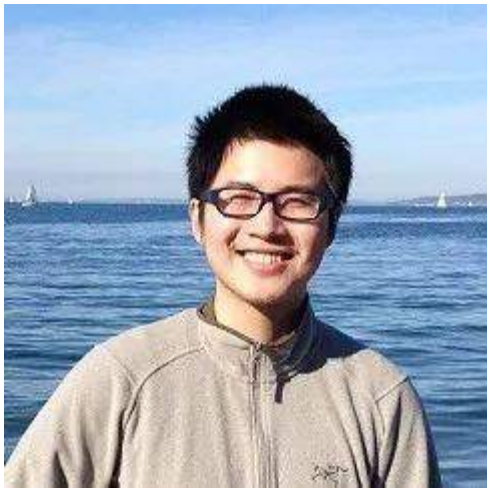
3，李沐，2008 年本科毕业于上海交通大学计算机系，CMU 博士毕业，深度学习开源框架 MXNet 作者之一，曾在微软亚洲研究院担任实习生，在亚马逊就职。沐神有一本在线书籍《动手学深度学习》，另外现在有很多的群，算是做深度学习的普及工作贡献了。



代表性文章：

[1] Li M , Liu Z , Smola A J , et al. DiFacto - Distributed Factorization Machines[C]// Acm International Conference on Web Search & Data Mining. ACM, 2016.

4、陈天奇，本科毕业于上海交通大学 ACM 班，华盛顿大学计算机系博士生。深度学习编译器 TVM，SVDFeature，XGBoost，cxxnet 等作者，MxNet，DMLC 发起人之一。



代表性文章：

[1] MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, Zheng Zhang LearningSys at Neural Information Processing Systems 2015

[2] TVM: An Automated End-to-End Optimizing Compiler for Deep Learning Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, Arvind Krishnamurthy

5、韩松，本科毕业于清华大学后，博士毕业于斯坦福大学，深鉴科技联合创始人之一，2016 年 ICLR 最佳论文 deep compression 论文一作。就放深鉴科技四个创始人的照片吧，都是青年才俊。

代表性文章：

[1] Han S , Kang J , Mao H , et al. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA[J]. 2016.

[2] Han S, Mao H, Dally W J, et al. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding[J]. international conference on learning representations, 2016.

3 总结

AI 领域不管是老前辈还是后起之秀真的太多了，写这一篇文章的目地不仅是给初学者们作一个简单的介绍，更是自勉。就算不能成为他们那样牛逼的人，也要有一颗见贤思齐，不断提升自己的斗志。

【AI 初识境】从 3 次人工智能潮起潮落说起

专栏《AI 白身境》已经完结，本篇开始这个系列的第二个专栏《AI 初识境》。所谓初识，就是对相关技术有基本了解，掌握了基本的使用方法。

第一篇，我们就来说说人工智能的简单历史，通常来说业界公认包括三个重要的历史阶段。

作者|言有三

1 什么是智能

智能，即 Intelligence，那什么是智能呢？按照中国古代思想家荀子在《荀子·正名篇》的说法：“所以知之在人者谓之知，知有所合谓之智。所以能之在人者谓之能，能有所合谓之能”。

其中的“智”指进行认识活动的某些心理特点，“能”则指进行实际活动的某些心理特点。我的理解就是，**能是智的具体表达形式**，当然也有学者将智和能作为一个整体的，《论衡·实知篇》中就认为**人才就是具有一定智能水平的人**。

更加科学的定义是在霍华德·加德纳的多元智能理论中，它将人类的智能分成七种能力：

- (1) 语言 (Verbal/Linguistic)
- (2) 逻辑 (Logical/Mathematical)
- (3) 空间 (Visual/Spatial)
- (4) 肢体运作 (Bodily/Kinesthetic)
- (5) 音乐 (Musical/Rhythmic)
- (6) 人际 (Inter-personal/Social)
- (7) 内省 (Intra-personal/Introspective)

这基本覆盖了现在人工智能的研究领域，包括计算机视觉，语音识别，自然语言处理等。

从生物学上来说智能体是有两种的，有生命的和没有生命的，人显然就是有生命的，而人工智能，自然研究的是没有生命体的，但是能够表现出生命体的智能的，因此才叫**“人工”**智能，或者理解为人造智能吧。

2 图灵与机器智能

科技发展到一个节点后，就有人思考如何用真正的生命体外的东西来模拟人类的智能，从模拟大脑的运作方式开始显然是最合适的。

在 1943 年的时候麦卡洛克-皮茨(MuCulloch 和 Pitts)就提出了 MP 模型，即最早的基于阈值逻辑的神经网络模型，用于模拟人脑神经元，它已经是感知器的原型了，开创了人工神经网络研究的时代。

当然，人工智能是一个比神经网络更广的范畴，它的理论先驱代表性人物是天才科学家**图灵**。

艾伦·图灵在 1950 年的论文《Computing Machinery and Intelligence》中提出了图灵测试，在图灵测试中，人类测试者与一台机器被隔开，一个人类测试员通过文字向被测试者随意提问。进行多次测试后，如果测试者不能确定出被测试者是人还是机器，那么这台机器就通过了测试，并被认为具有人类智能，它在数十年间被当作机器智能的测试标准，深刻影响了人工智能的发展。

当然图灵测试不需要限于对话，其他的一些人类活动也可以。目前在很多的领域里机器已经通过图灵测试，比如下棋。

很多学者认为，图灵在 1948 年的时候在英国国家物理实验室内部做过的一个题为“智能机器”的报告是更早的机器智能的源头，在这个报告中提出了“embodied intelligence”和 “disembobied intelligence”，即肉体智能和无肉体智能。肉体智能指的是需要身体来完成活动的，比如人形机器人，而无肉体智能则可以脱离形体，比如下棋，进行语言对话等。但是由于报告是保密的，因此我们还是认为 1950 年的图灵测试是真正意义上提出机器智能概念。

为了纪念图灵的贡献，美国计算机协会在 1966 年设立了图灵奖，这成为了计算机科学领域的“诺贝尔奖”，图灵也被称为计算机科学之父、人工智能之父。

3 冯诺伊曼与类脑计算

图灵提出了机器智能的概念，那怎么实现呢？现在大家都知道了，使用计算机，或者更通用的说法是电脑。

冯·诺依曼（John von Neumann）正是计算机之父。在智能这个研究课题上，图灵走理论路线，而冯·诺伊曼更偏重工程路线，它们在普林斯顿大学期间有接触，比图灵大 10 岁的冯诺伊曼对前者还有一定的知遇之恩。

冯诺伊曼的贡献很多，这里我们只说其在计算机方面的贡献，它对世界上第一台电子计算机**ENIAC**（电子数字积分计算机）的设计提出过建议，这体现在 1945 年他牵头起草的 EDVAC（Electronic Discrete Variable Automatic Computer）中，翻译过来是“存储程序通用电子计算机方案”。这对计算机的设计有决定性的影响，特别是计算机的结构，包括**存储程序以及二进制编码**等。这个报告中最核心的概念是“存储程序(Stored Program)”，老冯将其原创给予了图灵，认为存储程序其实就是通用图灵机，可见老冯不止是研究做得好，人品更是棒。

所谓存储程序：就是把运算程序存在机器的存储器中，程序设计员只需要在存储器中寻找运算指令，机器就会自行完成计算，这样，就不必对每个问题都重新编程，从而加快了程序设计的流程。

当然在 EDVAC 中还包括了**随机寻址**以及**寄存器**等原创思想。鉴于冯·诺依曼在发明电子计算机中所起的关键性作用，他被誉为“计算机之父”。冯诺伊曼的研究被整理在《计算机和人脑》中，感兴趣的读者可以去读中译本。

4 达特茅斯会议

图灵和冯·诺伊曼为机器智能奠基，而人工智能正式成为一个学科被广泛研究，应该起源于 1956 年的**达特茅斯会议**，会议的参与者包括了达特茅斯学院的**约翰·麦卡锡**

（John McCarthy）、哈佛大学的**马文·闵斯基**（Marvin Minsky，人工智能与认知学专家）、贝尔电话实验室的**克劳德·香农**（Claude Shannon，信息论的创始人）、**艾伦·纽厄尔**（Allen Newell，计算机科学家）、**赫伯特·西蒙**（Herbert Simon，诺贝尔经济学奖得主），塞弗里奇(Oliver Selfridge)等科学家，在两个多月的会议中它们讨论了用机器来模仿人类学习以及其他方面的智能这个课题。

虽然会议最后没有什么实质性的结论，但是会议正式提出了一个新学科， 即**“Artificial Intelligence”**，翻译过来就是人工智能。可以说一个学科的正式诞生，才是为公众所知的开始。

我们再来说一下参会的这些人和人工智能的关系。

1. 约翰·麦卡锡(John McCarthy)，开发了程序语言 Lisp(List Processing language)，成为第一个最流行的 AI 研究程序语言，他还提出了计算机分时(time-sharing)的概念。正是约翰·麦卡锡提出了“Artificial Intelligence”这个概念(虽然他自己老年说是听来的但是最后也无法证实来源)，因此也被称为“人工智能之父”(我觉得人工智能之母更加合适，毕竟图灵才是思想之源，而麦卡锡同志是一个养育者)，他在 1971 年获得了图灵奖。

2. 马文·闵斯基，他在 1951 年 24 岁的时候和迪恩·埃德蒙(Dean Edmunds)在普林斯顿大学建立了第一个神经网络机器 SNARC(Stochastic Neural Analog

ReinforcementCalculator)，这是第一个人工神经网络，用 3000 个真空管模拟出 40 个神经元的神经信号传递，由于他的一系列奠基贡献，在 1969 年获得了图灵奖。

3. 克劳德·香农(Claude Shannon)，在 1950 年发表了《Programming a Computer for Playing Chess》，这篇论文第一次开始关注计算机象棋程序的开发。作为信息科学领域的鼻祖，与图灵并驾齐驱的香农自然是不需要图灵奖了，毕竟还有通讯领域的最高奖香农奖。

4. 赫伯特·西蒙(Herbert Simon)和艾伦·纽厄尔(Allen Newell)，他们在 1956 年开发了"LogicTheorist"，是世界上第一个 AI 项目，证明了在罗素和怀特海的数学教科书《数学原理》第二章的 52 个定理中的 38 个，并找到了比教材中更加优美的证明，这开创了名为“搜索推理”的方法，这是人工智能的符号派的代表性成果。他们在 1975 年一起获得了图灵奖。

之后研究 AI 的一些科学家也获得了图灵奖，1994 年 Edward Feigenbaum)和 Raj Reddy、2010 年 Leslie Valiant、2011 年 Judea Pearl。正是有了这些科学家的头脑风暴，才有了今天的这门学科，改变了人类的生活方式，向他们致敬。

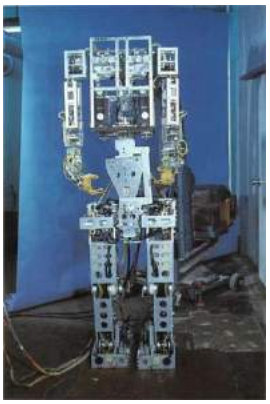
5 第一次浪潮(1956-1974)

在人工智能学科诞生后，赫伯特·西蒙(Herbert Simon)乐观地预测 20 年内诞生完全智能的机器，当然这没有发生，但是带来了第一个人工智能的研究热潮。

1963 年，美国国防部高级研究计划局(DARPA)给麻省理工学院、卡内基梅隆大学的人工智能研究組投入了 200 万美元的研究经费，开启了 Project MAC(Mathematics and Computation)，这个项目是麻省理工学院计算机科学与人工智能实验室的前身，培养了最早期的计算机科学与人工智能人才，也有了一些成果。

在 1964～1966 年间约瑟夫·维森班(loseph Weizenbaum)开发了 ELIZA，这是第一个自然语言对话程序，通过简单的模式匹配和对话规则进行任何主题的英文对话。不过他开发程序只是为了证明机器与人的交流很肤浅，后来成为了批判人工智能研究大军的一员。

早稻田大学在 1967 年到 1972 年间则发明了第一个人形机器人 Wabot-1，不仅可以进行简单的对话，还能够完成在室内走动和抓取物体的动作，这就涉及到了计算机视觉的研究。并且在 1980 年更新到了二代版本 Wabot-2，增加了阅读乐谱和演奏普通难度的电子琴的功能。研发该机器人的加藤一郎也被赋予**“人形机器人之父”**。



从 70 年代开始，由于计算能力有限，而科学家一开始的预测又过于乐观，导致研究和期望产生了巨大的落差，公众热情和投资削减，70 年代中期进入第一次人工智能的研究进入低谷。

6 第二次浪潮(1980-1987)

专家系统和人工神经网络的兴起，带来了第二波浪潮。

所谓专家系统，即基于特定的规则来回答特定领域的问题的程序，在 1964 年费根鲍姆(Edward Feigenbaum)等人在斯坦福大学研究了第一个专家系统 DENDRAL，能够自动做决策，解决有机化学问题，因此他也被成为**“专家系统之父”**。

在 1970 年，斯坦福大学的科学家们开发了专家系统 MYCIN，通过 600 多条人工编写的规则识别引发严重传染病的细菌，推荐抗生素。

在 1980 年，卡内基梅隆大学开发了 XCON 程序，这是一套基于规则开发的专家系统，帮助迪吉多公司的客户自动选择计算机组件，为其每年节省了 4000 万美元。

在巨大的商业价值的刺激下，工业界又兴起了对人工智能的热情。在这样的背景下，1982 年日本通商产业省启动了“第五代计算机”计划，期望通过大规模的并行计算来建造

通用人工智能平台，不过 10 年间耗资 500 亿日元后还是未能达到预期目标。休伊特(Carl Hewitt)认为大部分五代机的工作只是试着用逻辑程序去解决其他手段已经解决的问题。

1984 在年度 AAAI 会议上，罗杰·单克(Roger Schank)和马文·明斯基(Marvin Minsky)发出警告，认为“Al 寒冬”已经来临，Al 泡沫很快就会破灭，与此同时 AI 投资与研究资金也减少，正如 70 年代出现的事情一样。

但是另一方面，深度学习的前身，人工神经网络却取得了革命性的进展，在 1986 年戴维·鲁梅哈特(David Rumelhart)，杰弗里·辛顿(Geoffrey Hinton) 等人推广了由保罗·韦尔博斯(Paul Werbos) 发明的**反向传播算法(BP 算法)**，使得大规模神经网络训练成为可能。反向传播算法的出现，使得神经网络隐藏层可以学习到数据输入的有效表达，这就是神经网络乃至深度学习的核心思想。那时候的神经网络就好比上个世纪 90 年代的互联网，是一种时尚潮流。

虽然因为当时的计算机性能的限制，未能取得工业级的应用，但是这一次的蓄力，为第三次的兴起和爆发奠定了基础。

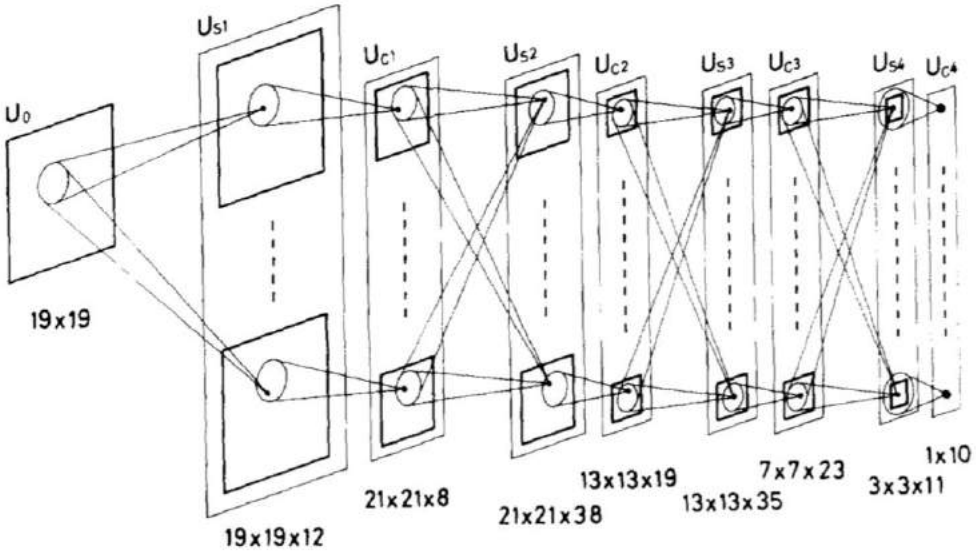
7 第二次浪潮后的蓄力

在第二次低谷之后，科学家们显然更加理智，当然也是因为其他学科暂时吸引了他们的注意力，包括**统计学习理论，支持向量机，概率图模型**等，这些方法带来了传统的机器学习方法的理论研究和应用，虽然机器学习在 1957 年就被阿瑟·萨缪尔(Arthur Samuel)提出。

在这一段时间里，统计学习类的机器学习算法就是人工智能的代表，但是由于它本身是一门数据驱动的应用学科，没有达到人工智能那样的广泛，因此大家不再叫人工智能，也降低了对它的期望，利用这些方法来做一些更加实际的问题，研究和应用方向也覆盖了计算机视觉到语音等等。

不过，在以 SVM 为代表的传统机器学习方法数十年间光芒的背后，却仍然有一股力量在蓄力。

在 1980 年 neocognitron 被提出，这是第一个真正意义上的级联卷积神经网络，虽然具体的卷积方式和今天的 CNN 还有一定的区别。



同一时期 RNN 也被提出并使用。

1989 年 Yann LeCun 与 AT&T 贝尔实验室的其它研究人员将反向传播算法应用于多层神经网络，并在 1998 年提出了稳定可商业应用的卷积神经网络模型 LeNet-5，成为了深层卷积神经网络模型的**“Hello World”**。虽然它不是第一次使用卷积神经网络，但是它的出现意味着将神经网络商用的可能性被验证。

1997 年赛普·霍克赖特(Sepp Hochrei ter))和于尔根·施密德胡伯(JurgenSchmidhuber)提出了 LSTM(长短期记忆)，用于识别手写笔迹和语音。

2001 年斯皮尔伯格拍摄的电影《人工智能》上映，这是一部非常具有里程碑意义的人工智能启蒙电影，所以我也将其放在这里讲述。

2006 年杰弗里·辛顿(Geoffrey Hinton)等人在 science 期刊上发表了论文“Reducing the dimensionality of data with neural networks”，揭开了新的训练深层神经网络算

法的序幕，通常这被认为是第三次浪潮的发源，但其实彼时还远远没有引起学界的重视，至少工业界几乎全然不知。

很快斯坦福的李飞飞等人开始研究 Image Net 项目，这是一个大型的开源数据库，在 2009 年正式发布，超过 1000 万数据，两万多个类别。2010 年 Image Net 大规模视觉识别挑战赛(ILSVCR)开始举办，从此揭开了计算机视觉研究的新序幕。

8 第三次浪潮

虽然 2006 年 Hinton 在 science 上发表的文章有足够的学术意义，但是第三次浪潮的兴起还是要从工业界发生的具有足够震撼力的标志性事件开始算起。

验证新技术是否足够好的最好方法就是和其他方法以及人类进行 PK。

2011 年 IBM 开发的自然语言问答计算机**“沃森”**在益智类综艺节目“危险边缘”中击败两名前人类冠军。前两轮与对手打平，而最后一集沃森打败了最高奖金得主布拉德·鲁特尔和连胜纪录保持者肯·詹宁斯，人们惊呼，机器也会思考了吗？

2012 年，在计算机视觉领域的竞赛 Image Net 中，新一代卷积神经网络 **AlexNet**，以提升 10%的错误率的进步力压第二名以 SIFT+FV, LBP+FV, GIST+FV, CSIFT+FV 等组合特征的算法。从此人类设计的特征再也不是机器自主学习特征的对手。虽然在此之前语音识别以及取得了足够大的进展，但是彼时语音识别仍然未能商用而没有那么被人所知。

2016 年 Google 的 **AlphaGo** 以 4:1 的成绩战胜了世界围棋冠军李世石。一年后，AlphaGoMaster 与人类实时排名第一的棋手柯洁对决，最终连胜三盘。而新一代 AlphaGo Zero 利用自我对抗迅速自学围棋，并以 100：0 的成绩完胜前代版本。自此 AI 下棋再无敌手。

此后，以深度学习为代表的技术，引领了当下的热潮。此所谓第三次潮起，会不会潮落不知道，我知道的是现在正在潮中。

9 总结

所谓读史使人明智，在正式开始学习之前，认真了解一下关键性的技术节点，了解符号派和统计派之争，了解前辈们的故事是很有必要的，建议读一读相关的书籍资料。

【AI 初识境】从头理解神经网络-内行与外行的分水岭

这是专栏《AI 初识境》的第 2 篇文章。所谓初识，就是对相关技术有基本了解，掌握了基本的使用方法。这一篇，我们就说说神经网络基础，包括简单历史，核心原理与技术。

作者|言有三

1 人工智能符号派与统计派

上一次我们讲述了人工智能的完整历史，自从图灵提出了“机器智能”，达特茅斯会议提出“人工智能”学科后，研究人员就开始活跃起来。

正所谓有人的地方就有江湖，人工智能的研究派系主要就是两大阵营。

第一大阵营，被称为**符号派**，他们用统计逻辑和符号系统来研究人工智能，采用自顶向下的研究思路，也就是要先看懂人工智能是什么，再去一步一步实现它。符号派用人工智能技术来证明公式，发明专家系统，代表人物有纽厄尔(Newell)、西蒙(Simon)等。

第二大阵营是**统计派**，现在的深度学习就属于这一派，这一派研究问题的方法就是仿造大脑，采用自底向上的思路。通过一个一个简单逻辑的搭建，搭建成一个足够复杂的系统后，系统就自然会有智能了。代表人物新时代的深度学习三大巨头 Yoshua Bengio，Hinton、LeCun。

应该说，符号派的研究方法最早也更加直观，典型的例子是专家系统，它的缺陷也非常明显，必须先对所研究的问题有完整的理解，然后再通过知识编码，这对于达到一定复杂程度的问题来说根本不可能实现，不是所有知识都能一下子说出来原因的。就好比在图像中识别一只猫，到底怎样的一幅图片才是猫图，传统的图像描述算子就很难完全定义，保证覆盖到各类场景。

统计派的思路就更加务实，摆事实讲道理，通过统计学习的方法来归纳出知识，不再完全由人工干预，天然和大数据是契合的，这也是成功的必然性。

有学者一生站队在一派，也有的摇摆过，比如上次说到的马文·明斯基，就从最开始的统计派在第二次 AI 浪潮中转变成为了符号派，并成为了第二次低谷的重要推手。

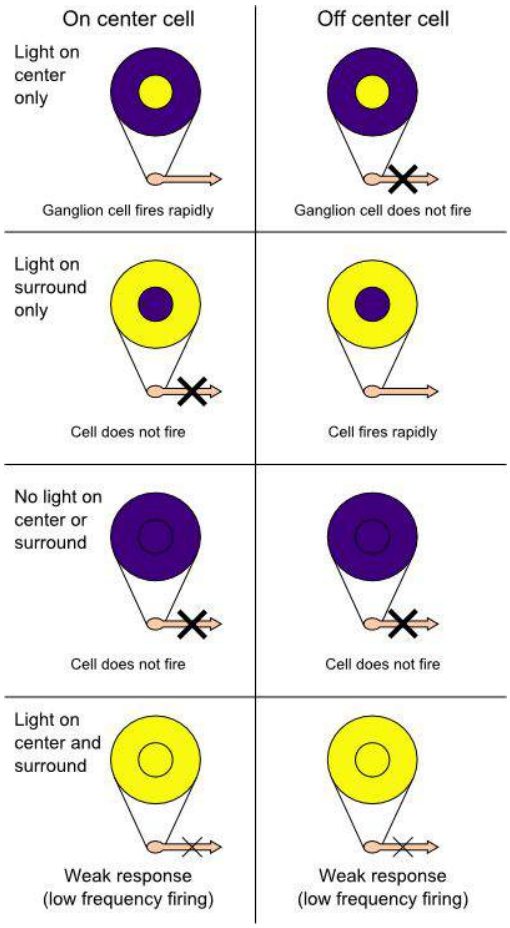
anyway，现在是统计学派大放异彩的时候。

2 人脑与视觉感知机制

前面说了统计派是采用了模仿大脑的方式，人类花了亿万年的时间来进化，那生物的大脑究竟是如何处理视觉信息呢？

大脑的基本感知单元就是神经元，一个神经元所影响的刺激区域就叫做**神经元的感受野**，即 receptive field，不同神经元感受野的大小和性质都不同。

感受野的研究来自于美国神经科学家哈特兰（Keffer Hartline）和匈牙利裔美国神经科学家库夫勒（Stephen W. Kuffler），1953 年他们发现猫视网膜神经节细胞的感受野具有同心圆结构。



如上图包括了两种感受野，第一种感受野**由作用强的中心兴奋区域和作用较弱但面积更大的周边抑制区域构成了一个同心圆结构**，称为 On 型感受野，第二类感受野是由中心抑制和周边兴奋的同心圆构成，称为 Off 型感受野。当刺激 On 型感受野中心时，细胞就兴奋，如第一列第一排的图。刺激周边时，细胞就抑制，如第一列第二排的图，Off 型图则反之。

不过尽管明白了感受野，视觉感知的机制仍然没有被得到更深刻地理解，直到视觉功能柱的发现。

加拿大神经生理学家 David Hunter Hubel 和瑞典神经科学家 Torsten Nils Wiesel 在 20 世纪 50 年代和 60 年代开始研究视觉机制，他们将图像投射到屏幕上，

测量神经元活动的电线插入猫的大脑，通过固定猫的头部来控制视网膜上的成像，测试生物细胞对线条、直角、边缘线等图形的反应。

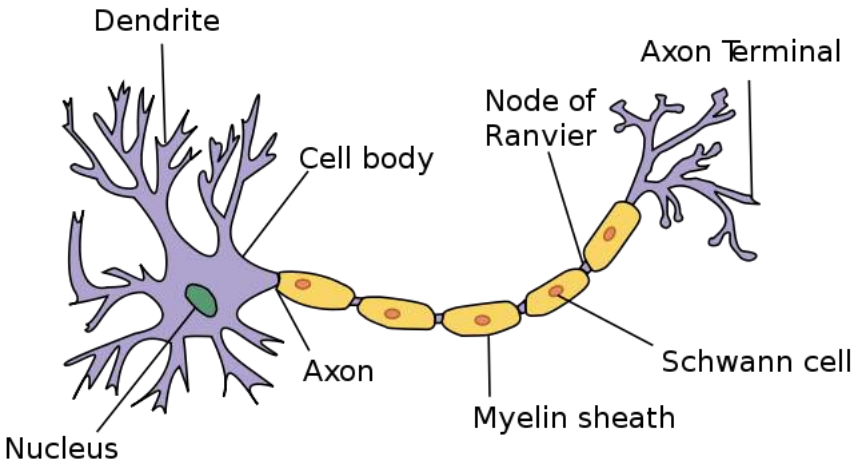
研究显示有些细胞对某些处在一个角度上的线条、垂直线条、直角或者明显的边缘线，都有特别的反应，这就是绝大多数视皮层细胞都具有的强烈的**方位选择性**。不仅如此，要引起这个细胞反应，直线的朝向还只能落在一个很小的角度范围里，也就是该细胞的感受野内。这样以上两个研究就统一起来了。

从 1960 年到 1980 年，两人合作了 20 多年，细致科学地研究了人眼视觉的机制，因此他们被认为是现代视觉科学之父，并于 1981 年一起获得了诺贝尔生理学与医学奖。

这一项研究，直接催生了 David Marr 的视觉分层理论，也催生了计算机视觉学科，更是卷积神经网络的发源。

3 MP 模型

在生物学家，计算机科学家的共同努力下，大家摸清了人脑神经元的工作机制。



神经元之间相互连接，当某一神经元处于“兴奋”状态时，其相连神经元的电位将发生改变，若神经元电位改变量超过了一定的数值（也称为阈值），则被激活处于“兴奋状态”，向下一级连接的神经元继续传递电位改变信息。信息从一个神经元以电传导的方式跨过细胞之间的联结(即突触)，传给另一个神经元，最终使肌肉收缩或腺体分泌。

神经元通过突触的连接使数目众多的神经元组成比其他系统复杂得多的神经系统。从神经元的结构特性和生物功能可以得出结论：**神经元是一个多输入单输出的信息处理单元，并且对信息的处理是非线性的。**

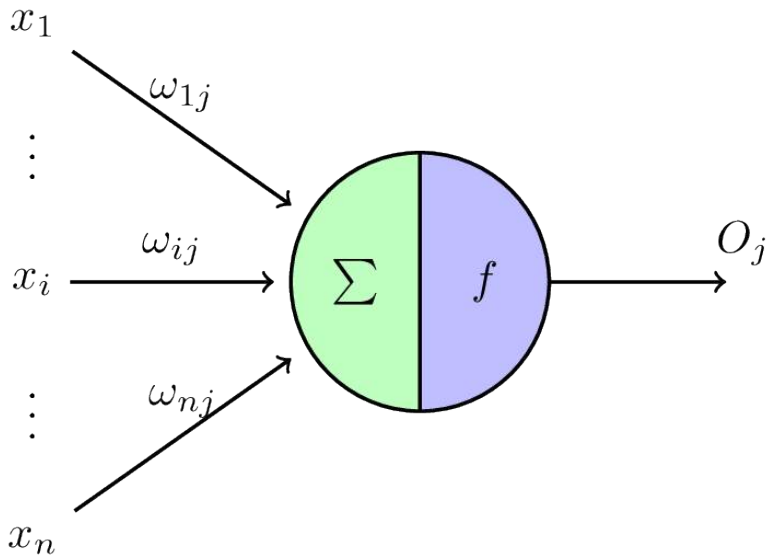
在这个基础上，就有科学家产生了模拟神经网络的想法。1943 年 McCulloch 和 Pitts 提出了 **MP 模型**，这是一种基于阈值逻辑的算法创造的神经网络计算模型，由固定的结构和权重组成。

在 MP 模型中，某个神经元接受来自其余多个神经元的传递信号，多个输入与对应连接权重相乘后输入该神经元进行求和，再与神经元预设的阈值进行比较，最后通过函数产生神经元输出。每一个神经元均为多输入单输出的信息处理单元，具有空间整合特性和阈值特性。

其实 MP 模型就已经是感知器的原型了，只是没有真正在计算机上实现而已。

4 感知机

感知机（Perceptron）是 Frank Rosenblatt 在 1957 年提出的概念，其结构与 MP 模型类似，一般被视为最简单的人工神经网络，也作为二元线性分类器被广泛使用。通常情况下指单层的人工神经网络，以区别于多层感知机（Multilayer Perceptron）。尽管感知机结构简单，但能够学习并解决较复杂问题。

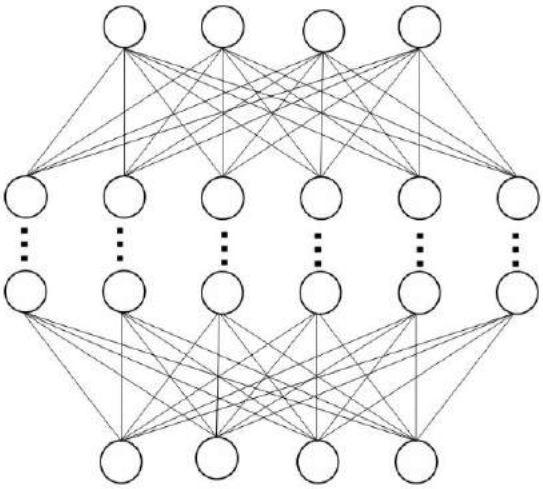


假设我们有一个 n 维输入的单层感知机，x1 至 xn 为 n 维输入向量的各个分量，w1j 至 wnj 为各个输入分量连接到感知机的权量（或称权值），theta 为阈值，f 为激活函数（又称为激励函数或传递函数），o 为标量输出。理想的激活函数通常为阶跃函数或者 sigmoid 函数。感知机的输出是输入向量 x 与权重向量 w 求得内积后，经激活函数 f 所得到的标量。

单层感知器类似一个逻辑回归模型，可以做线性分类任务，但是不能做更复杂的任务。第二次 AI 浪潮中马文·明斯基在其著作中证明了感知器本质上是一种线性模型，只能处理线性分类问题，就连最简单的 XOR（亦或）问题都无法正确解决。作为人工智能领域的开创者之一，这一声明也直接或间接促使神经网络的研究陷入了近 20 年的停滞。

5 多层感知器与反向传播

不过就算在低谷期，1974 年哈佛大学的 Paul Werbos 仍然证明增加一个网络层，利用反向传播算法可以搞定 XOR 问题。到了后来 Rummelhart，McClelland 以及 Hinton 在 1986 年正式在多层感知器(MLP)中使用 BP 算法，采用 Sigmoid 进行非线性映射，有效解决了非线性分类和学习的问题。



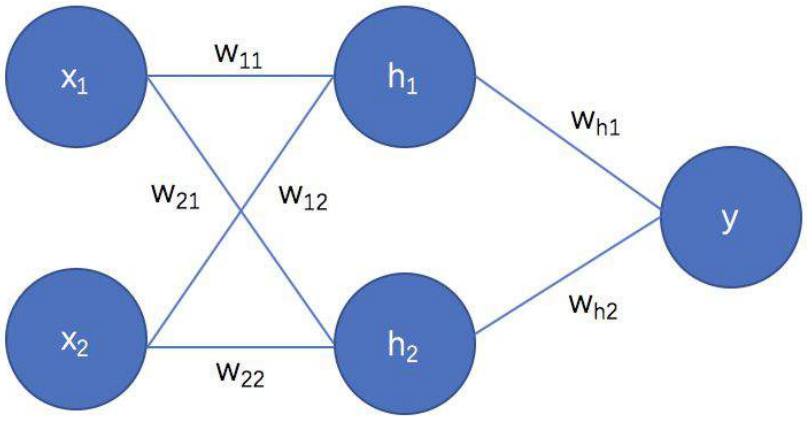
多层感知机（Multi-Layer Perceptron）是由单层感知机推广而来，最主要的特点是有多个神经元层。一般将 MLP 的第一层称为输入层，中间的层为隐藏层，最后一层为输出层。MLP 并没有规定隐藏层的数量，因此可以根据实际处理需求选择合适的隐藏层层数，且对于隐藏层和输出层中每层神经元的个数也没有限制。

多层感知机的关键问题在于如何训练其中各层间的连接权值，

方法有一些不过大家最熟知的就是**反向传播 BP 算法**了。

反向传播算法的具体推导涉及大量的公式，实在不适合在公众里写，因此我们就不写了，大家随便找一本书都能找到资料，勤快的可以自己推导一遍。

这里给大家一个实际的案例来体会：



输出为 y，损失函数为 E。

$$y = h_1 w_{h1} + h_2 w_{h2} = x_1 w_{11} + x_2 w_{12} + x_2 w_{21} + x_2 w_{22}$$

$$\mathrm{E} = \frac{1}{2}(y - t)^2$$

假如某一时刻值如下：

x_1=1，x_2=-1，w_{11}=0.1，w_{21}=-0.1，w_{12}=-0.1
w_{22}=0.1，w_{h1}=0.8，w_{h2}=0.9，t=0

$$h_1 = w_{11} x_1 + w_{12} x_2 = 0.2$$
$$h_2 = w_{21} x_1 + w_{22} x_2 = -0.2$$
$$y = h_1 w_{h1} + h_2 w_{h2} = -0.02$$

那么我们可以计算 E 对 Wh1 的误差传播值为：

$$\frac{\partial \mathrm{E}}{\partial w_{h1}} = \frac{\partial \mathrm{E}}{\partial y} \frac{\partial y}{\partial w_{h1}} = (y - t) h_1 = -0.004$$

下次更新 Wh1 这个参数的时候就可以采用：

$$w_{h1} = w_{h1} - \eta \frac{\partial \mathrm{E}}{\partial w_{h1}}$$

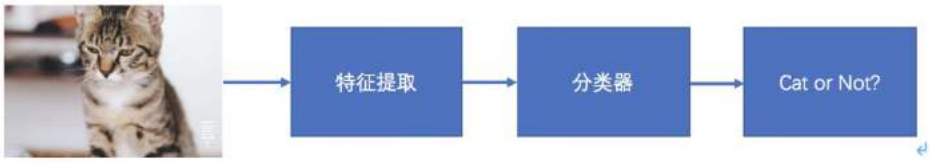
η 就是学习率了，原理就是这样，一层一层推导下去就行了。

反向传播算法让多层感知器，或者说传统的全连接神经网络有了训练的手段，引发了神经网络的第二次热潮，虽然为期不长，毕竟当时算力和数据都很有限，但是全连接神经网络总算是正式起来了。

6 全连接神经网络 2 大缺陷

传统的 BP 神经网络在 20 世纪 80 年代左右流行，但是很快因为 SVM 等核方法的诞生而黯然失色。这是因为传统的 BP 神经网络有几个重大的缺陷。

(1) 首先是原理上的缺陷：BP 神经网络仍然是有监督的传统机器学习方法，遵循着以下思路。



也就是说，不过是在最后面将SVM或者其他分类器换成神经网络，在大部分情况下其实没有什么优势，甚至增加了问题的复杂度。

提取的特征虽然是研究者们经过反复实验证明有效的特征，但仍然会一定程度上丢失了图像中的结构信息，从而丢失了一些对旋转扭曲等的不变性。而且要求输入的大小是固定的。为了学习到如偏移等微小的变化，需要有足够多的参数和足够多丰富的样本，最终学习到的权重，很可能在不同的位置处还会有非常相似的权重。

有人可能会说，直接把图像作为输入而不提取特征行不行？请接着往下看。

(2) 再一个就是结构上的缺陷：参数巨多，丢失空间信息。

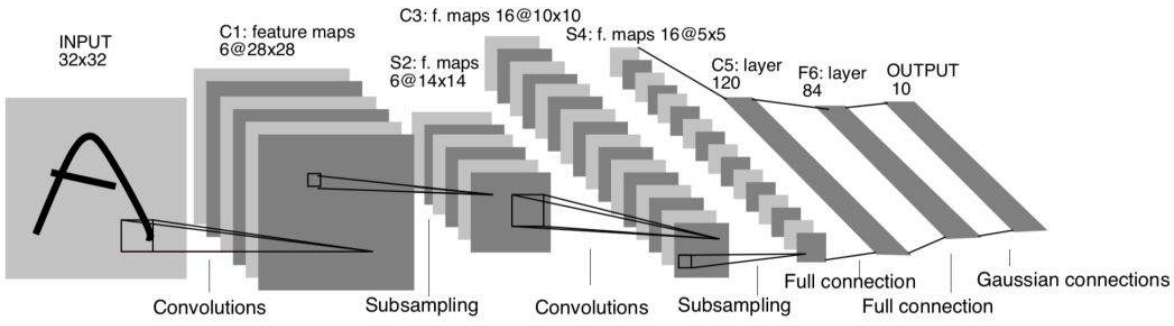
全连接神经网络从 BP 算法提出开始，发展于 90 年代，那时候的计算机属于 CPU 时代，根本就无法撑起海量参数的计算。

如果一个隐藏层特征图像大小为 100*100，输入层的特征图像大小为 100*100，这意味着学习这一层需要 100*100*100*100=10^8 的参数。如果以 32 位的浮点数进行存储，就需要 4*108 的字节的存储量，约等于 400MB 的参数量。仅仅这样的一个网络层，其模型参数量已经超过了 AlexNet 网络的参数量，而 100*100 的特征图像分辨率，已经低于很多任务能够成功解决的下限。除了计算过程中需要存储的海量的参数，还有海量的计算，这些都超过了当时硬件的能力，因此大大限制了网络的大小，尤其是对于一些大的图像输入。

7 为什么是卷积神经网络

不管是历史局限性也好，还是神经网络有种种毛病，总之 80 年代后的 20 年间它不是主流。

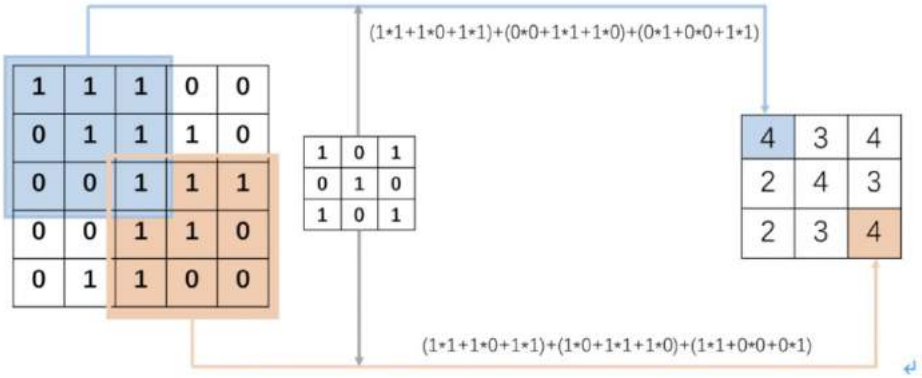
不过在上个世纪 90 年代研究神经网络的学者们没有停止，经典的诸如 LeNet5 这样的网络被提出。



为什么是卷积神经网络呢？首先自然是要知道什么是卷积神经网络。

1，什么是卷积？

卷积在工程和数学上有非常多的应用，在信号处理领域中，任意一个线性系统的输出，就是输入信号和系统激励函数的卷积。放到数字图像处理领域，卷积操作一般指图像领域的二维卷积。



一个二维卷积的案例如上，在图像上滑动，取与卷积核大小相等的区域，逐像素做乘法然后相加。

例如原始图像大小是 5*5，卷积核大小是 3*3。首先卷积核与原始图像左上角 3*3 对应位置的元素相乘求和，得到的数值作为结果矩阵第一行第一列的元素值，然后卷积核向右移动一个单位（即步长 stride 为 1），与原始图像前三行第 2、3、4 列所对应位置的元素分别相乘并求和，得到的数值作为结果矩阵第一行第二列的元素值，以此类推。

故卷积就是：一个核矩阵在一个原始矩阵上从上往下、从左往右扫描，每次扫描都得到一个结果，将所有结果组合到一起得到一个新的结果矩阵。

注意这里我们不区分卷积和互相关，它们的区别只在于权重算子是否进行了翻转。之所以不重视，是因为在机器学习中，卷积核是否翻转，并不影响算法学习。

2，为什么要用卷积来学习呢？

图像都是用方形矩阵来表达的，学习的本质就是要抽象出特征，以边缘检测为例。它就是识别数字图像中亮度变化明显的点，这些点连接起来往往是物体的边缘。

传统的边缘检测常用的方法包括一阶和二阶导数法，本质上都是利用一个卷积核在原图上进行滑动，只是其中各个位置的系数不同，比如 3*3 的 sobel 算子计算 x 方向的梯度幅度，使用的就是下面的卷积核算子。

$$\begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix}$$

如果要用 sobel 算子完成一次完整的边缘检测，就要同时检测 x 方向和 y 方向，然后进行融合。这就是两个通道的卷积，先用两个卷积核进行通道内的信息提取，再进行通道间的信息融合。

这就是卷积提取特征的本质，而所有基于卷积神经网络来学习的图像算法，都是通过不断的卷积来进行特征的抽象，直到实现网络的目标。

3，卷积神经网络的优势在哪？

前面说了全连接神经网络的原理和结构上的缺陷，而这正好是卷积的优势。

(1) 首先是学习原理上的改进，卷积神经网络不再是有监督学习了，不需要从图像中提取特征，而是直接从原始图像数据进行学习，这样可以最大程度的防止信息在还没有进入网络之前就丢失。

(2) 另一方面是学习方式的改进。前面说了全连接神经网络一层的结果是与上一层的节点全部连接的，100×100 的图像，如果隐藏层也是同样大小（100*100 个）的神经元，光是一层网络，就已经有 10^8 个参数。要优化和存储这样的参数量，是无法想象的，所以经典的神经网络，基本上隐藏层在一两层左右。而卷积神经网络某一层的结点，只与上一层的一个图像块相连。

用于产生同一个图像中各个空间位置像素的卷积核是同一个，这就是所谓的**权值共享**。对于与全连接层同样多的隐藏层，假如每个神经元只和输入 10×10 的局部 patch 相连接，且卷积核移动步长为 10，则参数为：100×100×10×10，降低了 2 个数量级。

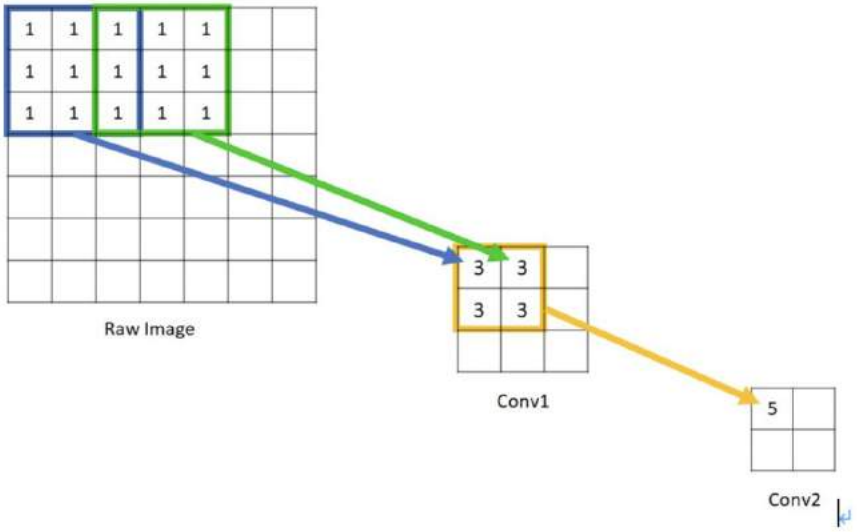
又能更好的学习，参数又低，卷积神经网络当然是可以成功了。

8 卷积神经网络的核心基础概念

在卷积神经网络中，有几个重要的基本概念是需要注意的，这在网络结构的设计中至关重要。

(1) 感受野

直观上讲，感受野就是视觉感受区域的大小。在卷积神经网络中，感受野是CNN中的某一层输出结果的一个元素对应输入层的一个映射，即 feature map 上的一个点所对应的输入图上的区域，具体示例如下图所示。



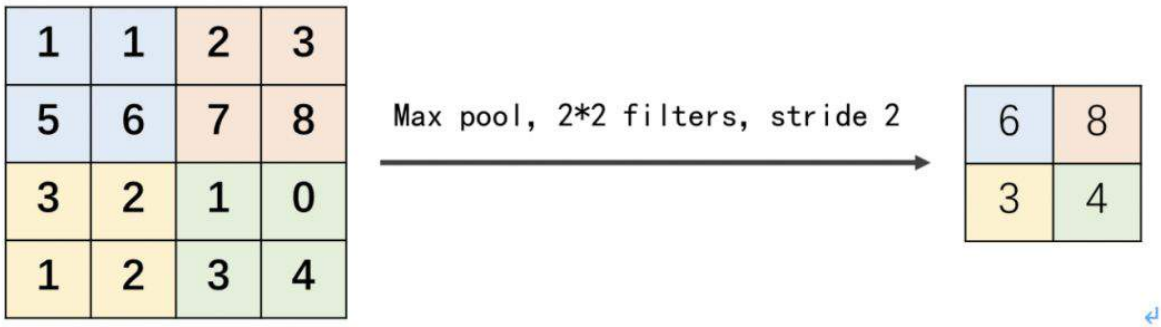
如果一个神经元的大小是受到上层N*N的神经元的区域的影响，那么就可以说，该神经元的感受野是 N*N，因为它反映了 N*N 区域的信息。在上图 conv2 中的像素点 5，是由 conv1 的 2*2 的区域计算得来，而该 2*2 区域，又是由 raw image 中 5*5 的区域计算而来，所以，该像素的感受野是 5*5。可以看出感受野越大，得到的全局信息越多。在物体分割，目标检测中这是非常重要的一个参数。

(2) 池化

有了感受野再来解释池化（pooling）也很简单，上图的raw image 到conv1，再到conv2，图像越来越小。每过一级就相当于一次降采样，这就是池化。池化可以通过步长不为 1 的卷积实现，也可以通过 pool 直接插值采样实现，本质上没有区别，只是权重不同。

通过卷积获得了特征之后，下一步则是用这些特征去做分类。理论上讲，人们可以把所有解析出来的特征关联到一个分类器，例如 softmax 分类器，但计算量非常大，并且极易出现过度拟合（over-fitting）。而池化层则可以对输入的特征图进行压缩，一方面使特征图变小，简化网络计算复杂度；一方面进行特征压缩，提取主要特征。

一般而言池化操作的池化窗口都是不重叠的，所以池化窗口的大小等于步长 stride。如下图所示，采用一个大小为2*2的池化窗口，max pooling是在每一个区域中寻找最大值，这里的stride=2，最终在原特征图中提取主要特征得到右图。



除此之外，还有卷积核的大小，卷积的步长，通道的边界填充值等等，都是很好理解的基本概念，有一些以后会进行更加详细的剖析。

这一次就说到这，希望大家从此对神经网络和卷积神经网络(CNN)的基础概念不再有疑问。

9 总结

模仿大自然可能就是人类一直以来学习和发明的最重要的方法，神经网络就是这么一个神奇的东西。作为 IT 技术人员，不管从事不从事 AI 领域，都应该稍微了解一下这个方向。

【AI 初识境】近 20 年深度学习在图像领域的重要进展节点

这是专栏《AI 初识境》的第 3 篇文章。所谓初识，就是对相关技术有基本了解，掌握了基本的使用方法。这是本系列的最后一篇非技术文章，我们总结一下深度学习技术在图像领域的重要历史性节点，本来打算语音，自然语言处理一起的，文章太长以后再谈。

作者&编辑 | 言有三

1 前深度学习时代

从早期的全连接神经网络到卷积神经网络 CNN，跨度超过半个世纪，我们在上一期文章中进行过回顾，大家感兴趣的可以回过头去看。

几个重要的节点是：

1943 年：MP 模型的提出。

1960～1980 年：视觉机制的发现。

1979 年：Neocognitron 的提出，卷积神经网络的萌芽。

1986 年：反向传播算法被用于神经网络的优化并开始流行，同期动量算法提出被用于加速 SGD。

1990 年：TDNN 模型，卷积神经网络被用于语音识别。

1992 年：Max-pooling 被提出，此后成为卷积神经网络标准组件。

1997 年：LSTM 被提出，促进了语音，自然语言处理等领域等发展。

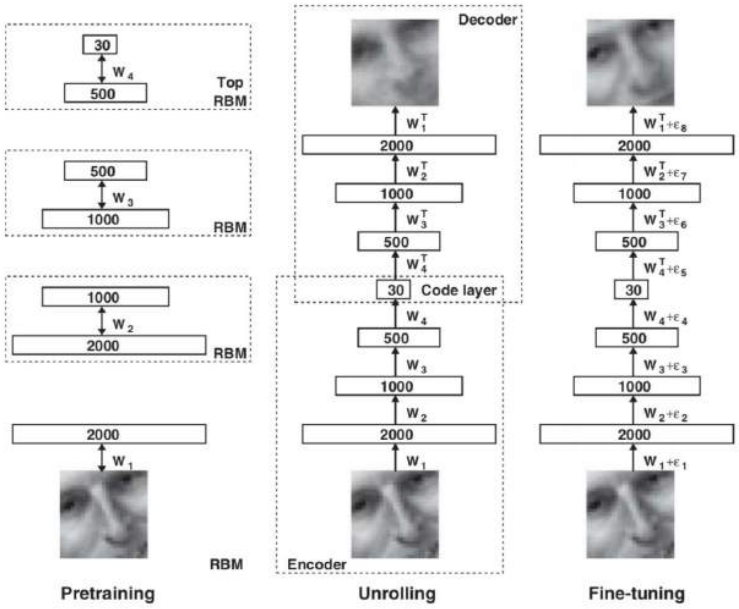
1998 年：LeNet5 和 MNIST 数据集被提出和整理，两者可以说各自是卷积神经网络和图像数据集的“Hello World”，总会被拿出来说一说。

所谓深度学习，是以人工神经网络为基本架构的特征学习方法，涵盖监督学习，无监督学习，半监督学习，增强学习等，模型结构以**卷积神经网络**为代表，它不仅被用于图像，也被用于语音，自然语言处理等各种领域。

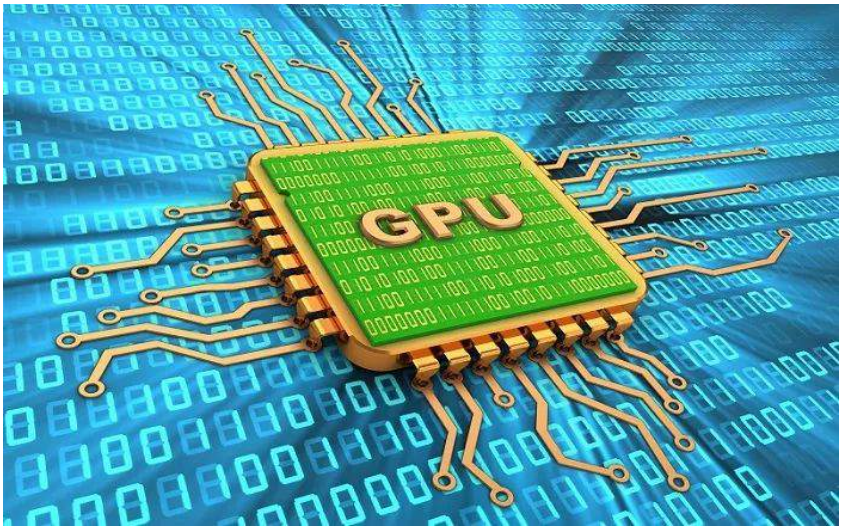
2 深度学习时代

以 2006 年为分水岭，下面尽量挑重点的，在学术界和工业界有重大意义，同时又广为人知的来说。

2006 年 Hinton 等人在 science 期刊上发表了论文“Reducing the dimensionality of data with neural networks”，揭开了新的训练深层神经网络算法的序幕。利用无监督的 RBM 网络来进行预训练，进行图像的降维，取得比 PCA 更好的结果，**通常这被认为是深度学习兴起的开篇**。



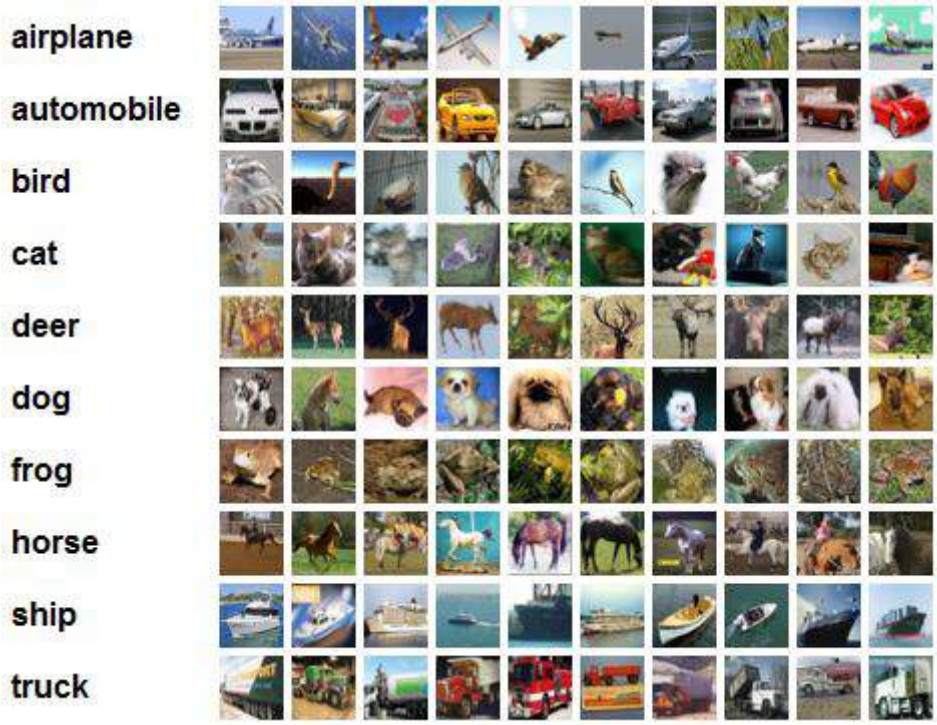
2006 年，**NVIDIA 推出 CUDA**，GPU 被用于训练卷积神经网络，是当时的 CPU 的训练速度的四倍。到现在，GPU 是研发强大算法必备的条件，这也是大公司屡屡取得突破而小公司只能亦步亦趋跟随的一个很重要的原因。NVIDIA 的 GeForce 系列，搞深度学习的谁还没有呢？



2006～2009 年，在图像 MNIST 数据集，语音 TIMIT 数据集以及一些垂直领域的小比赛比如 TRECVID 也取得了不错的进展，但是还算不上突破性的，所以也不怎么为人所知。

2009 年，**CIFAR10 和 CIFAR100 数据集**被整理。由于 MNIST 是一个灰度图像数据集，而大部分现实的任务为彩色图像，所以 Alex Krizhevsky 等学者从 TinyImage 数据集中整理出了 CIFAR10 和 CIFAR100。与 MNIST 一样 CIFAR10 数据集也有 60000 张图像，不过图像为彩色。图像大小是 32×32，分为 10 个类，每类 6000 张图。其中 50000 张用于训练，另外 10000 用于测试。CIFAR100 则分为 100 个类，每一类 600 张图像。

这两个数据集与 MNIST 一样，在评测方法时非常常见。



2009 年，**ImageNet 数据集**被整理，并于次年开始每年举办一次比赛。ImageNet 数据集总共有 1400 多万幅图片，涵盖 2 万多个类别，为计算机视觉领域做出了巨大的贡献，至今我们仍然使用着 Imagenet 来评估算法，以及预训练其他任务的模型。

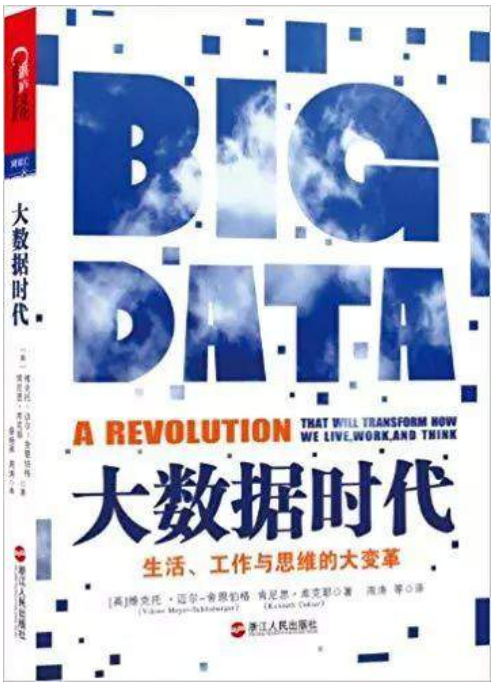
2009 年前后几年时间，属于**融汇贯通各种技术，数据和装备**，典型的蓄力阶段，辅以小数据集和若干比赛的突破。

2011 年，CNN 以 0.56%的错误率赢得了 IJCNN 2011 比赛并超过了人眼，这是一场交通标志的识别比赛，研究者开始对深度学习在自动驾驶中的应用前景展现出浓厚的兴趣，毕竟在上个世纪 90 年代无人车的研究就已经开始了。现在无人车是非常大的一个应用前景。

2011 年，Glorot 等人提出 **ReLU 激活函数**，有效地抑制了深层网络的梯度消失问题，现在最好的激活函数都是来自于 ReLU 家族，简单而有效。

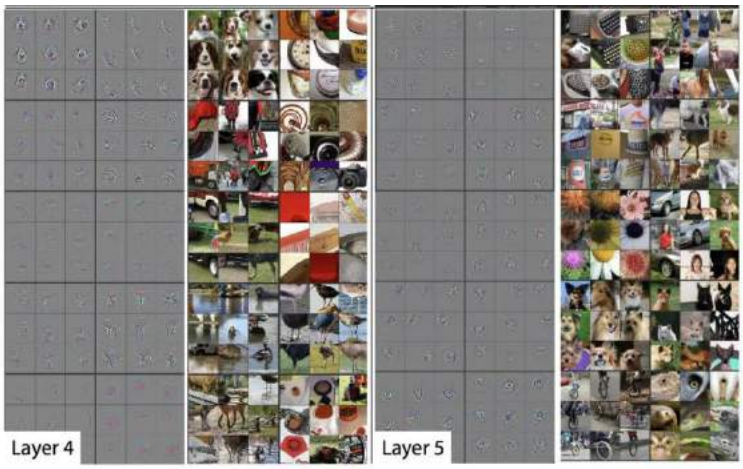
2012 年，经典书籍**《大数据时代》**出版，作者维克托?迈尔?舍恩伯格在书中指出大数据时代来了，**我们应该放弃对因果关系的追求，而关注相关关系，从“为什么”开始转变到“是什么”**，这不就是统计学习人工智能学派的基础工具深度学习最擅长做的吗。

也就是从那个时候开始，人们大喊，大数据来了，一时之间，数据科学家，数据挖掘工程师成为热门。

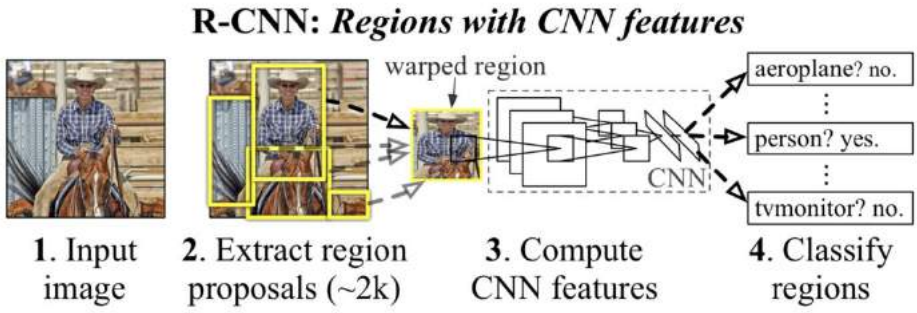


2012 年，Hinton 的学生 Alex Krizhevsky 提出 **AlexNet 网络**，以低于第 2 名 10%的错误率赢得了 ImageNet 竞赛。当时 Alex Krizhevsky 使用了两块显卡 GTX580，花了 6 天时间才训练出 AlexNet，我相信如果有更多的资源，AlexNet 一定是一个更好的 AlexNet。

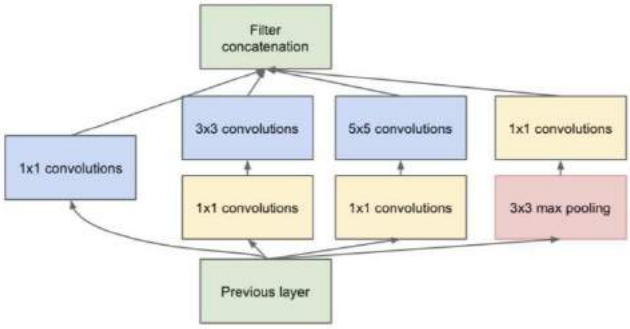
2013 年 Hinton 的学生 Zeiler 和 Fergus 在研究中利用反卷积技术引入了神经网络的可视化，提出了 zfnet，对网络的中间特征层进行了可视化，为研究人员检验不同特征激活及其与输入空间的关系成为了可能，慢慢地大家也开始都关注起深度学习的作用机制。



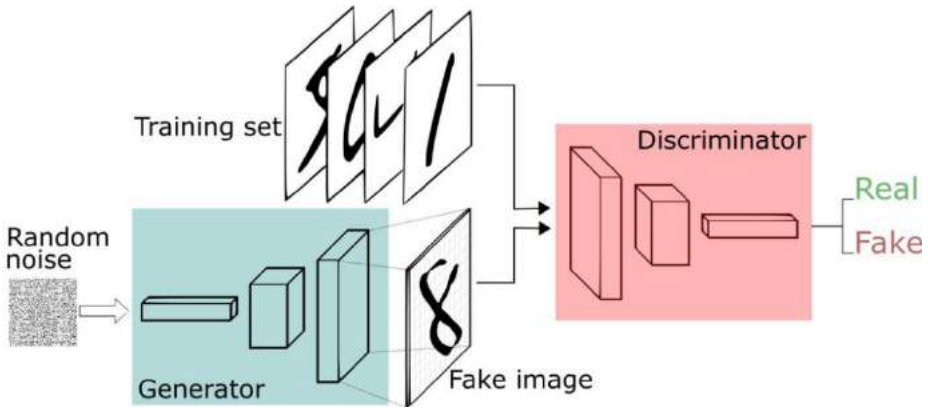
2013 年，Ross Girshick 等人提出了**目标检测模型 RCNN**，开创了 CNN 用于目标检测的基准之一。随后研究者针对该系列提出 Fast RCNN，Faster RCNN 等等。



2014 年，**GoogLeNet 和 VGGNet** 分别被提出，获得 ImageNet 分类赛的冠亚军。VGGNet 很好的展示了如何在先前网络架构的基础上通过简单地增加网络层数和深度就可以提高网络的性能，GoogleNet 模型架构则提出了 Inception 结构，拓宽神经的宽度，成为了计算效率较高的深层模型基准之一。



2014 年，无监督学习网络 **GAN** 横空出世，独立成了一个新的研究方向，被 LeCun 誉为下一代深度学习，此后 GAN 在各大领域，尤其是图像领域不断“建功立业”，并与各类 CNN 网络结构进行了融合。



2015 年，**ResNet** 获得了 ImageNet2012 分类任务冠军，以 3.57%的错误率表现超过了人类的识别水平，并以 152 层的网络架构创造了新的模型记录，自此残差连接在 CNN 的设计中随处可见。

2015 年，**全卷积网络 Fully Convolutional Networks** 被提出用于图像分割，自此图像分割领域也即迎来大爆发。

2014 年，Google 启动 **AlphaGo** 的研究，2015 年 10 月 AlphaGo 击败欧洲围棋冠军樊麾成为第一个无需让子即可击败围棋职业棋手的计算机围棋程序。2016 年 3 月，AlphaGo 在一场世界瞩目的比赛中 4:1 击败顶尖职业棋手李世石，2017 年 5 月 23 至 27 日在乌镇围棋峰会上，AlphaGo 和世界第一棋手柯洁比试全胜。

AlphaGo 的成功，对人工智能的普及工作意义非常深远，让不仅是从业者，外行人也开始领略到人工智能的强大，而背后就有卷积神经网络的功劳。

此后便是卷积神经网络在计算机视觉各大领域攻城略地，无往而不胜。关于都有哪些方向，可以参考这个。

【AI 白身境】一文览尽计算机视觉研究方向

而各种各样的卷积神经网络架构被提出，可参见我们之前的一个总结。

【完结】总结 12 大 CNN 主流模型架构设计思想

从上面的这些历史可以看出，很多重要的研究其实都是同一时期出现，而最后为人所知虽然有先后的顺序，但是金子迟早会发光。

这也不仅让我们要思考，接下来几年里大放异彩的，是现在哪些刚刚初出茅庐却还没有名噪天下的东西呢？

3 总结

重要的节点通常都承前启后，不管是作为谈资，还是设身处地地站在当时的节点来思考一番，都是受益良多的。

【AI 初识境】激活函数：从人工设计到自动搜索

这是专栏《AI 初识境》的第 4 篇文章。所谓初识，就是对相关技术有基本了解，掌握了基本的使用方法。在神经网络中，有一个看似不起眼但是非常重要的概念，那就是激活函数。激活函数模型固然理解起来简单，但是也经历了从人工设计到自动探索的长足发展历程。

作者&编辑 | 言有三

1 无处不在的激活函数

我们都知道人工神经网络是用于模拟神经元的，那么提起激活函数，自然是要从那里去挖掘原因。

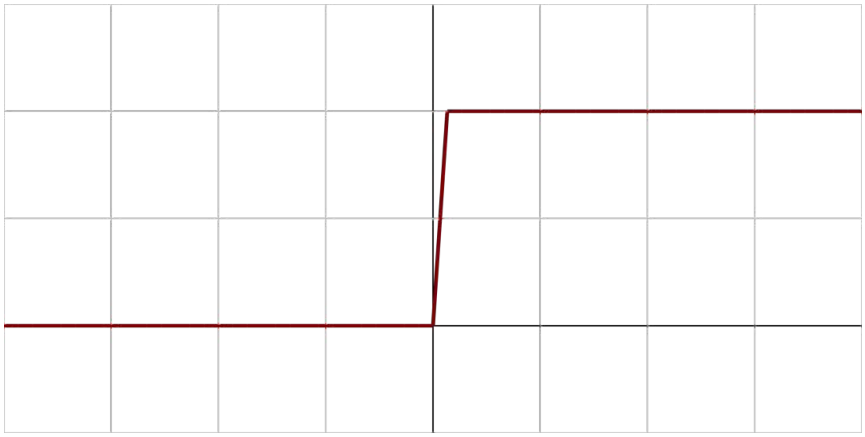
在正说深度学习中的激活函数之前，我想说**其实激活函数无处不在**。

(1) 上班需要激活函数。早上 10 点上班，你 8 点到也好，9 点到也好都一样，但是 10 点零 1 分到不行，性质就变了，考勤系统留下迟到的记录，全勤奖再无希望。

它的激活函数应该是这样的，x 是打卡时间。

$$f(x)=\begin{cases}1, & x<10点\\ 0, & x>10点\end{cases}$$

这是一个阶跃函数，类似于如下：

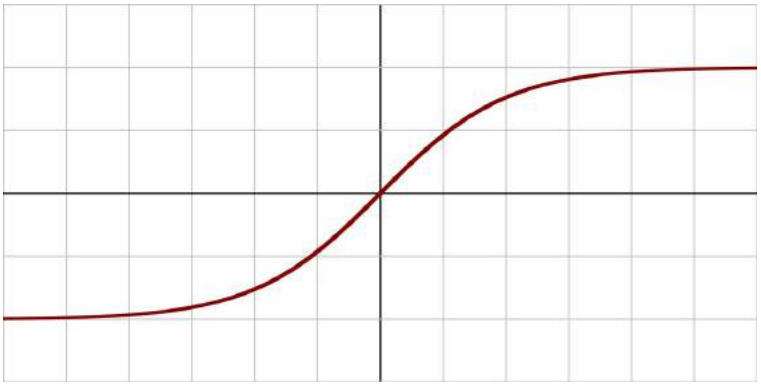


(2) 最近看上了一个跳槽过来的喜欢吃甜品的女同事，不过听说有男朋友，不过又听说好像正在慢慢闹分手。

那么如果要追这个女同事，什么时候送甜品能带来友情的升华？假如预判她和对象第 t 天后拜拜。

它的激活函数可能是这样的，x 是送甜品的日子。

$$f(x)=\frac{e^{x-t}-e^{-x+t}}{e^{x-t}+e^{-x+t}}$$

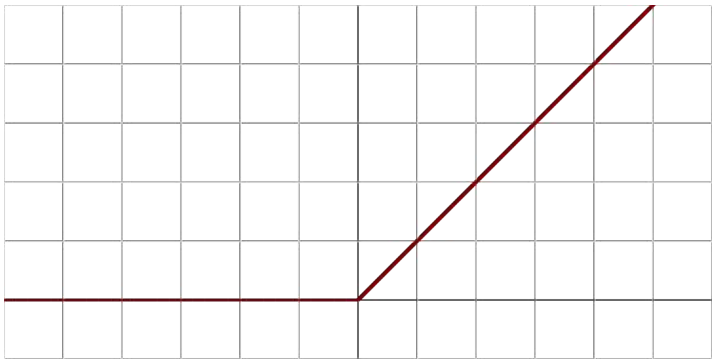


在刚分手的时候(也就是第 t 天，对应曲线斜率最大的地方)送甜品带来的好感激增度是最高的，再往后虽然随着相互之间越来越熟友谊持续升温，但是增长率下降了啊。而且到后来可能被其他人追走了，这个函数还只在一定期限内有效。

(3) 最近项目要加班，不过好在加班费是按小时(可以有分数)算的，那么当天的工资，就应该是这样算的。它的激活函数可能是这样的，x 是工作时长。

$$f(x)=\begin{cases}1000, & x<8\\ 1000+(x-8)\times 200, & x>8\end{cases}$$

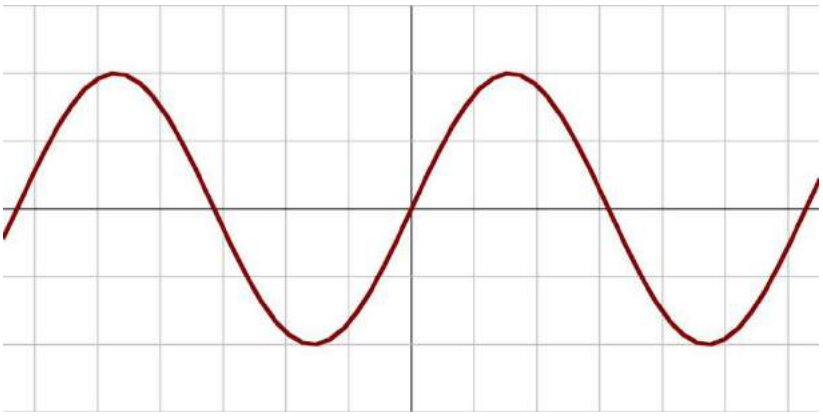
形状长这样，超过一个阈值后是线性增加的，低于阈值则是常量。



(4) 不是单身狗？OK 你是有老婆的人，那么下班回家陪老婆看电视总需要吧。不过到底陪不陪看，是不是陪就一定能得到老婆大人喜欢，这个可能是个周期性质的东西。

假如 x 是当天日历，那么激活函数可能是这样。

$$f(x)=\sin x$$



这么想想，是不是感觉激活函数无处不在，上面的这几种都是有正儿八经对应的激活函数的。

回转正题，之所以需要激活函数，从生物学上来说，是因为人脑的细胞接受刺激从而产生活动，首先需要一定的阈值，没有达到阈值，几乎没用。而不同的刺激产生的输出也是不同的，达到一定值后就饱和了，再加大也没用。

作为模拟人脑的人工神经网络，自然是需要某种机制来模拟这一种活动，这便是激活函数根本性的由来。

2 激活函数到底有什么用

一个复杂的神经网络，是有许多层的，其中最基本的单位便是神经元。一个线性神经元，输入 x 输出 y 的变换关系如下。

$$y = b + \sum_i x_i w_i$$

可以看到输出 y 与 x 是一个线性关系。如果再增加一层，把 y 作为中间层，输出为 z 呢？

如下：

$$y_j = b_j + \sum_i x_i w_{ji}$$

$$z = b' + \sum_j y_j w'_j = b' + \sum_j \left(b_j + \sum_i x_i w_{ji} \right) w'_j = b' + \sum_j (b_j w'_j) + \sum_j \left(\sum_i x_i w_{ji} w'_j \right) = b' + \sum_j (b_j w'_j) + \sum_i \left(\sum_j x_i w_{ji} w'_j \right)$$

可以看出，最终的输出 z 仍然与 x 是线性关系，也就是说这样堆叠下去，永远都是线性关系。

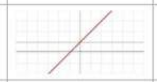
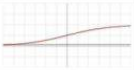
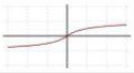
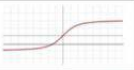
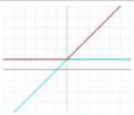
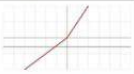
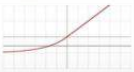
人们期望神经网络可以模拟任意的函数，怎么可能用一个线性函数来完成呢？所以才会在线性神经元的输出后添加非线性的函数，添加的越多，变换自然就越复杂了。

而不同的非线性映射函数的选择，就是激活函数的研究课题了。

3 各种激活函数

https://en.wikipedia.org/wiki/Activation_function

关于激活函数的种类，有三这一次就偷个懒，大家可以去 wiki 百科上面看，非常的详细，下面摘录其中的一些。

Name	Plot	Equation	Derivative (with respect to x)	Range
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$[1]	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$
ElliotSig[9][10][11] Softsign[12][13]		$f(x) = \frac{x}{1+\lvert x\rvert}$	$f'(x) = \frac{1}{(1+\lvert x\rvert)^2}$	$(-1, 1)$
Inverse square root unit (ISRU)[14]		$f(x) = \frac{x}{\sqrt{1+\alpha x^2}}$	$f'(x) = \left(\frac{1}{\sqrt{1+\alpha x^2}}\right)^3$	$\left(-\frac{1}{\sqrt{\alpha}}, \frac{1}{\sqrt{\alpha}}\right)$
Inverse square root linear unit (ISRLU)[14]		$f(x) = \begin{cases} \frac{x}{\sqrt{1+\alpha x^2}} & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \left(\frac{1}{\sqrt{1+\alpha x^2}}\right)^3 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$\left(-\frac{1}{\sqrt{\alpha}}, \infty\right)$
Square Nonlinearity (SQNL)[11]		$f(x) = \begin{cases} 1 & : x > 2.0 \\ x - \frac{x^2}{4} & : 0 \leq x \leq 2.0 \\ x + \frac{x^2}{4} & : -2.0 \leq x < 0 \\ -1 & : x < -2.0 \end{cases}$	$f'(x) = 1 \mp \frac{x}{2}$	$(-1, 1)$
Rectified linear unit (ReLU)[15]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$
Bipolar rectified linear unit (BReLU)[16]		$f(x_i) = \begin{cases} ReLU(x_i) & \text{if } i \bmod 2 = 0 \\ -ReLU(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$	$f'(x_i) = \begin{cases} ReLU'(x_i) & \text{if } i \bmod 2 = 0 \\ -ReLU'(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$	$(-\infty, \infty)$
Leaky rectified linear unit (Leaky ReLU)[17]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Parameteric rectified linear unit (PReLU)[18]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$[2]
Randomized leaky rectified linear unit (RReLU)[19]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \text{[3]} \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Exponential linear unit (ELU)[20]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$	$(-\alpha, \infty)$

这些人工设计的激活函数有这么多，那么什么激活函数最好，是 ReLU 吗？还是各类 ReLU 的变种(LReLU，PReLU，RReLU，ELU，SELU，GELU 等等)，Maxout，又或者是某大神自己在搞的很复杂的激活函数，这是没有答案的，只能说有一些通用的大家认可的结论，下面也只能覆盖到一些。

(1) sigmoid 和 tanh 激活函数。

为什么最早的时候大家用 sigmoid 函数呢？因为它不管输入处于多大的范围，输出是处于 0～1 的，机器学习里要解决的问题很多都是概率，用 sigmoid 不是很自然吗？

就算它有所谓的饱和问题导致梯度很小，那也是在函数的尾部才会发生，或者是在多级连乘之后才明显。所以很早期的比较浅的神经网络，用 sigmoid 没毛病，现在在 LSTM 这一类需要计算开关概率的网络中，sigmoid 仍然是很常见的。

那 tanh 函数又如何呢？它相比 sigmoid 来说，将输出映射到(-1,1)之间了，拓展了一倍的值域。激活有负值之后，网络的表达能力可以得到提升，但未必一定需要这么做的，因为权重本身是可以为负的，而在最早期的神经网络中，用模拟信号处理问题，甚至连权重都没有非负的，一样有效。不过一般来说 tanh 总不至于比 sigmoid 差的，它毕竟通过零点，输出期望不会漂移。

(2)ReLU 激活函数。

好处是很明显的。首先它简单，这个简单不仅在于导数恒定，更在于它将低于一定阈值的信号丢弃了。深度学习要解决的是工程问题，工程问题很多时候都是稀疏性的，往往简单的解决方案是最有效和稳定的。不过 ReLU 输出没有负数的问题确实有一定负作用，这也是其他方法对 ReLU 的改进空间所在。

(3)ReLU 的一大堆变种(LReLU，PReLU，RReLU，ELU，SELU，GELU 等等)。

我相信这些变种是有用的，但是我没怎么用过。不用因为是首先它们还没有表现出一定比 ReLU 强，在如今有 BN 等技术以及好的初始化方法后，ReLU 的缺点没有那么明显了。另一方面是，没时间去一个一个试，解决问题的过程中还有很多其他因素更加需要去探索。不过，还是建议大家去仔细了解一下的，用不用的着再说。

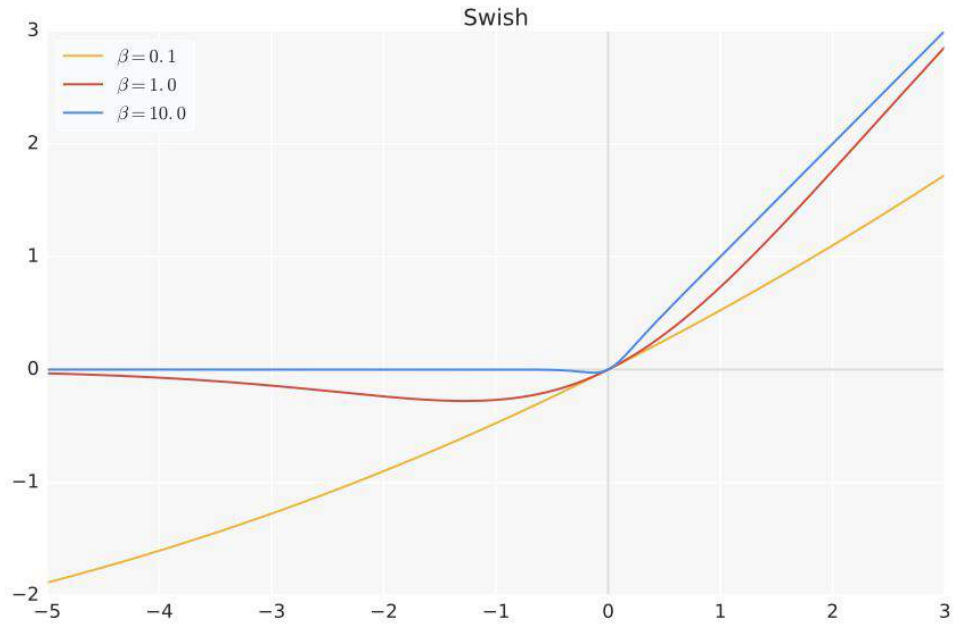
正因如此，在对 ReLU 改进的差不多之后，激活函数的人工设计就没有这么多热情了。

4 自动搜索

不过坑还没有填完，还是有人没有忘记这个问题的，比如谷歌。谷歌开了许多深度学习领域的自动化的工作，比如自动设计网络 NASNet，自动数据增强 AutoAugment 等工作，也做了自动搜索最优的激活函数的工作。

文[1]就在一系列**一元函数和二元函数组成的搜索空间**中，进行了比较细致的组合搜索实验。

结论是好用的激活函数都比较简单，不会超过两个基本函数的乘的组合。搜到了一些比 Relu 表现更好的函数，最好的是一个这样的函数：$x \cdot \sigma(\beta x)$，被称为 Swish，它在某个特定的参数下也和 ReLU 及其变种类似，看看图就知道了。



顺便说一下该方法做实验时的一元函数和二元函数的搜索空间：

- **Unary functions**: $x, -x, |x|, x^2, x^3, \sqrt{x}, \beta x, x+\beta, \log(|x|+\epsilon), \exp(x)\ \sin(x), \cos(x), \sinh(x), \cosh(x), \tanh(x), \sinh^{-1}(x), \tan^{-1}(x), \text{sinc}(x), \max(x,0), \min(x,0), \sigma(x), \log(1+\exp(x)), \exp(-x^2), \text{erf}(x), \beta$
- **Binary functions**: $x_1+x_2, x_1 \cdot x_2, x_1-x_2, \frac{x_1}{x_2+\epsilon}, \max(x_1,x_2), \min(x_1,x_2), \sigma(x_1)\cdot x_2, \exp(-\beta(x_1-x_2)^2), \exp(-\beta|x_1-x_2|), \beta x_1+(1-\beta)x_2$

已经覆盖我们能想到的一些简单的函数了。

类似地也有其他的研究人员通过遗传算法学习到一些新的激活函数，包括 **EliSH，HardEliSH[2]**，感兴趣的可以去看论文。

这个坑就挖给你了，还可以填。

[1] Ramachandran P, Zoph B, Le Q V. Searching for activation functions[J]. arXiv preprint arXiv:1710.05941, 2017.

[2] Basirat M , Roth P M . The Quest for the Golden Activation Function[J]. 2018.

[3] Nwankpa C , Ijomah W , Gachagan A , et al. Activation Functions: Comparison of trends in Practice and Research for Deep Learning[J]. 2018.

5 总结

深度学习各个维度的理论正处于全面被研究中，如果你想有所建树，那么必须要深入思考以前那些看似习以为常的东西，激活函数就是一个例子。

【AI 初识境】什么是深度学习成功的开始？参数初始化

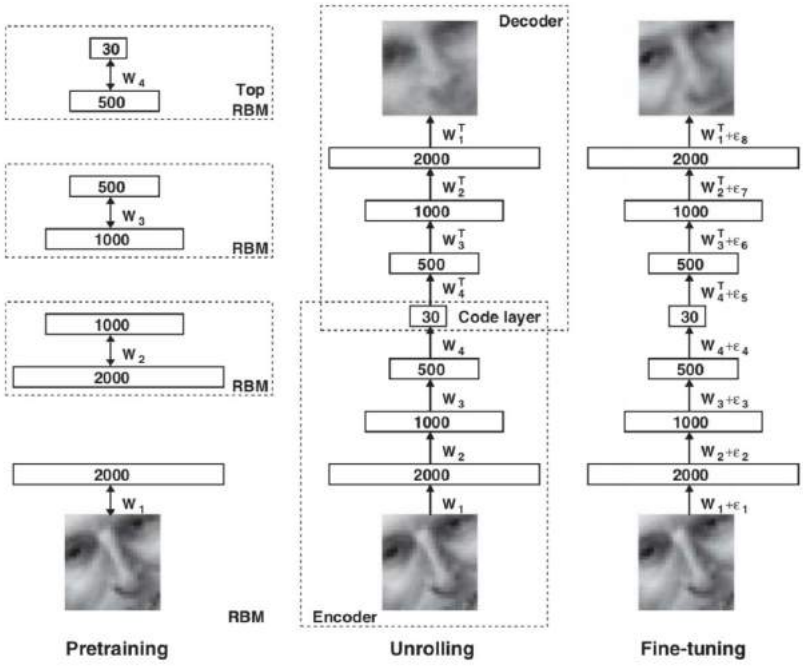
这是《AI 初识境》第 5 篇，这次我们说说初始化。所谓初识，就是对相关技术有基本了解，掌握了基本的使用方法。神经网络要优化一个非常复杂的非线性模型，而且基本没有全局最优解，初始化在其中扮演着非常重要的作用，尤其在没有 BN 等技术的早期，它直接影响模型能否收敛。可以说万事开头难，没有好的初始化的深度学习模型训练起来更难。

作者&编辑 | 言有三

1 初始化的重要性

2006 年 Hinton 等人在 science 期刊上发表了论文“Reducing the dimensionality of data with neural networks”，揭开了新的训练深层神经网络算法的序幕。

利用无监督的 RBM 网络来进行预训练，进行图像的降维，取得比 PCA 更好的结果，通常这被认为是**深度学习兴起的开篇**。



这么看来，是因为好的初始化方法的出现，才有了深层神经网络工程化落地的可能性。

好的初始化应该满足以下两个条件：

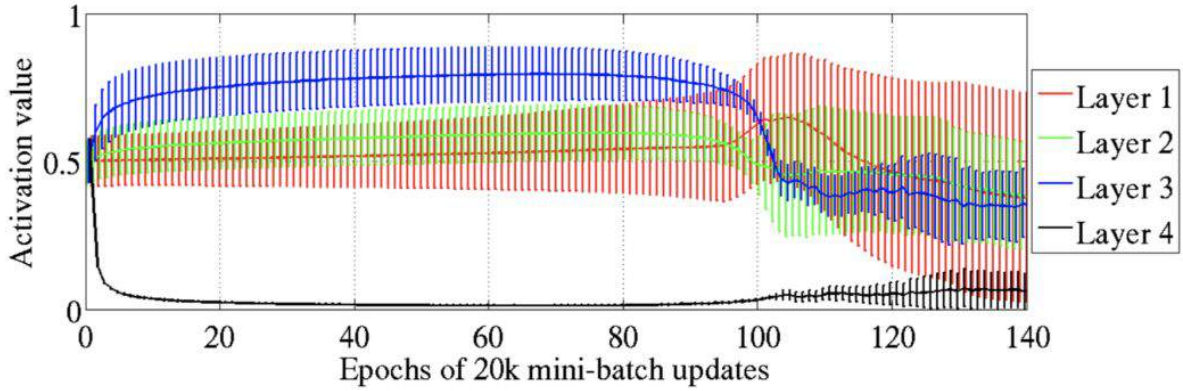
(1) 让神经元各层激活值不会出现饱和现象；

(2) 各层激活值也不能为 0。

也就是激活值不要太大，也不要太小，应该刚刚好，当然这还只是最基本的要求。

我们都知道在早期，sigmoid 激活函数是多层感知器模型的标配，上面这篇文章同样也是用 sigmoid 激活函数，没有那么多问题，是因为使用了预训练。

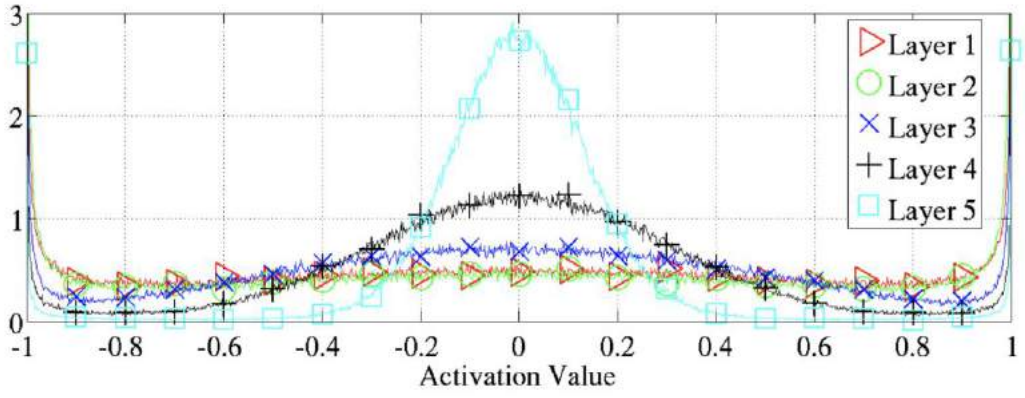
如果不使用预训练会如何？在 Xavier Glorot 和 Yoshua Bengio 提出 xavier 初始化方法的论文【1】中就对不同的激活函数使用不同的数据集做过实验。



上面是一个四层的神经网络在 sigmoid 函数激活下的训练过程，可以看到最深的 layer4 层刚开始的时候很快就进入了饱和区域(激活值很低)，其他几层则比较平稳，在训练的后期才能进行正常的更新。

为什么会这样呢？网络中有两类参数需要学习，一个是权重，一个是偏置。对于上面的结果作者们提出了一个假设，就是在网络的学习过程中，偏置项总是学的更快，网络真正的输出就是直接由 layer4 决定的，输出就是 softmax(b+Wh)。既然偏置项学的快，那 Wh 就没有这么重要了，所以激活值可以低一点。

解释虽然比较牵强，但是毕竟实验结果摆在那里，从 tanh 函数的激活值来看会更直观。



这个图有个特点是，0，1 和-1 的值都不少，而中间段的就比较少。在 0 值，逼近线性函数，它不能为网络的非线性能力作出贡献。对于 1，-1，则是饱和区，没有用。

2 常用的初始化方法

1、 全零初始化和随机初始化

如果神经元的权重被初始化为 0， 在第一次更新的时候，除了输出之外，所有的中间层的节点的值都为零。一般神经网络拥有对称的结构，那么在进行第一次误差反向传播时，更新后的网络参数将会相同，在下一次更新时，相同的网络参数学习提取不到有用的特征，因此深度学习模型都不会使用 0 初始化所有参数。

而随机初始化就是搞一些很小的值进行初始化，实验表明大了就容易饱和，小的就激活不动，再说了这个没技术含量，不必再讨论。

2.标准初始化

对于均匀分布，X～U(a,b)，概率密度函数等于：

$$f(x)=\begin{cases}\dfrac{1}{b-a}, & x\in(a,b)\\ 0, & \text{其他}\end{cases}$$

它的期望等于 0，方差等于(b-a)^2/12，如果 b=1，a=-1，就是 1/3。

下面我们首先计算一下，输出输入以及权重的方差关系公式：

$$y=\sum_{i}^{n}w_i x_i$$

$$\mathrm{Var}(y)=\mathrm{Var}(\sum_{i}^{n}w_i x_i)=\sum_{i}^{n}Var(w_i x_i)=\sum_{i}^{n}[E(w_i)]^2Var(x_i)+\sum_{i}^{n}[E(x_i)]^2Var(w_i)+Var(w_i x_i)$$

如果输入和权重都是零均值，那么

$$Var(y)=\sum_{i}^{n}Var(w_i x_i)=nVar(w)\sum_{i}^{n}Var(x_i)$$

如果我们希望每一层的激活值是稳定的，w 就应该用 n 的平方根进行归一化，n 为每个神经元的输入数量。

所以标准的初始化方法其权重参数就是以下分布：

$$(-1/\sqrt{n},1/\sqrt{n})$$

它保证了参数均值为 0，方差为常量 1/3，和网络的层数无关。

3.Xavier 初始化

首先有一个共识必须先提出：神经网络如果保持每层的信息流动是同一方差，那么会更加有利于优化。不然大家也不会去争先恐后地研究各种 normalization 方法。

不过，Xavier Glorot 认为还不够，应该增强这个条件，好的初始化应该使得**各层的激活值和梯度的方差在传播过程中保持一致，这个被称为 Glorot 条件。**

如果反向传播每层梯度保持近似的方差，则信息能反馈到各层。而前向传播激活值方差近似相等，有利于平稳地学习。

当然为了做到这一点，对激活函数也必须作出一些约定。

(1) 激活函数是线性的，至少在 0 点附近，而且导数为 1。
(2) 激活值关于 0 对称。

这两个都不适用于 sigmoid 函数和 ReLU 函数，而适合 tanh 函数。

要满足上面的两个条件，就是下面的式子。

$$\forall i, \quad n_i Var[W^i] = 1$$

$$\forall i, \quad n_{i+1} Var[W^i] = 1$$

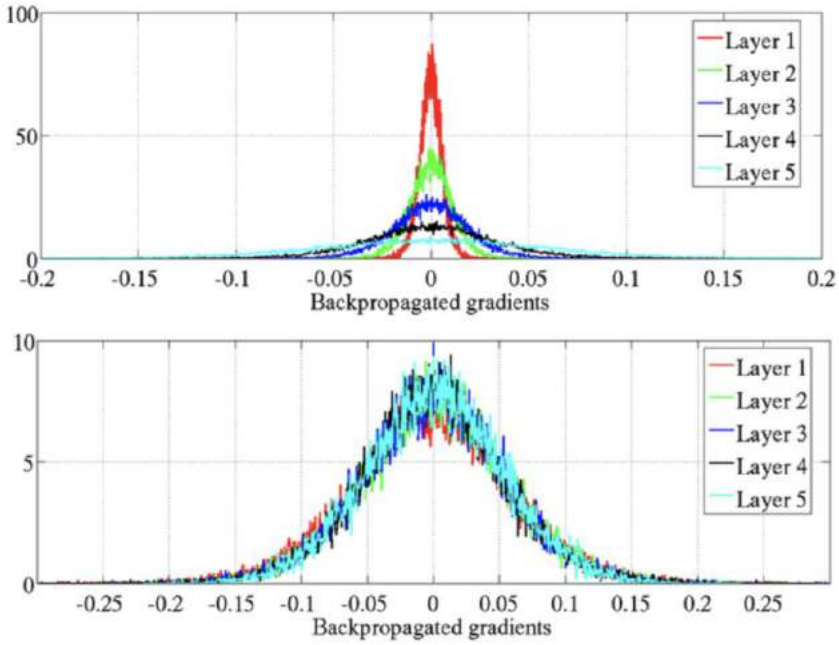
推导可以参考前面标准初始化的方法，这里的 ni，ni+1 分别就是输入和输出的神经元个数了，因为输入输出不相等，作为一种权衡，文中就建议使用输入和输出的均值来代替。

$$\forall i, \quad Var[W^i] = \frac{2}{n_i + n_{i+1}}$$

再带入前面的均匀分布的方差(b-a)^2/12，就得到了对于 tanh 函数的 xavier 初始化方法。

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}} \right] \qquad (16)$$

下面这两个图分别是标准初始化和 xavier 初始化带来的各层的反传梯度方差，可以看出 xavier 确实保持了一致性。



4. He 初始化

Xavier 初始化虽然美妙，但它是针对 tanh 函数设计的，而激活函数现在是 ReLU 的天下，ReLU 只有一半的激活，另一半是不激活的，所以前面的计算输入输出的方差的式子多了一个 1/2，如下。

$$\begin{aligned} Var[\Delta x_l] &= \hat{n}_l Var[w_l] Var[\Delta y_l] \\ &= \frac{1}{2}\hat{n}_l Var[w_l] Var[\Delta x_{l+1}]. \end{aligned}$$

因为这一次没有使用均匀初始化，而是使用了正态分布，所以对下面这个式子：

$$\frac{1}{2}\hat{n}_l Var[w_l] = 1, \quad \forall l.$$

需要的就是这样的正态分布。

$$W \sim G\left[0, \sqrt{\frac{2}{n}}\right]$$

综上，对于两大最经典的激活函数，各自有了对应的初始化方法。虽然后面还提出了一些其他的初始化方法，但是在我们这个系列中就不再详述了。

3 关于初始化的一些思考

初始化这个问题明显比较麻烦，不然大家也不会这么喜欢用 **pretrained 模型**了。

从前面我们可以看到，大家努力的方向有这么几个。

(1) 预训练啊。

机智地一比，甩锅给别人。

(2) 从激活函数入手，让梯度流动起来不要进入饱和区，则什么初始化咱们都可以接受。

这其实就要回到上次我们说的激活函数了，ReLU 系列的激活函数天生可以缓解这个问题，反过来，像何凯明等提出的方法，也是可以反哺激活函数 ReLU。

(3) 归一化，让每一层的输入输出的分布比较一致，降低学习难度。

回想一下，这不就是 BN 干的活吗？所以才会有了 BN 之后，初始化方法不再需要小心翼翼地选择。假如不用 BN，要解决这个问题有几个思路，我觉得分为两派。

首先**是理论派**，就是咱们从理论上分析出设计一个怎么样的函数是最合适的。

对于 Sigmoid 等函数，xavier 设计出了 xavier 初始化方法，对于 ReLU 函数，何凯明设计了 he 初始化方法。

在此之上，有研究者分别针对零点平滑的激活函数和零点不平滑的激活函数提出了统一的框架，见文【2】，比如对于 sigmoid，tanh 等函数，方差和导数的关系如此。

$$v^2 = \frac{1}{N(g'(0))^2(1+g(0)^2)},$$

然后是**实践派**，在训练的时候手动将权重归一化的，见文【3】，这就是向归一化方法靠拢了，下期咱们再讲。

[1] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks[C]//Proceedings of the thirteenth international conference on artificial intelligence and statistics. 2010: 249-256.

[2] Kumar S K. On weight initialization in deep neural networks[J]. arXiv preprint arXiv:1704.08863, 2017.

[3] Mishkin D, Matas J. All you need is a good init[J]. arXiv preprint arXiv:1511.06422, 2015.

4 总结

好的初始化方法就是赢在起跑线，不过现在的初始化方法也不是对什么数据集都有效，毕竟不同数据集的分布不同，咱们以后再谈。

【AI 初识境】深度学习模型中的 Normalization，你懂了多少？

这是《AI 初识境》第 6 篇，这次我们说说 Normalization。所谓初识，就是对相关技术有基本了解，掌握了基本的使用方法。数据经过归一化和标准化后可以加快梯度下降的求解速度，这就是 Batch Normalization 等技术非常流行的原因，它使得可以使用更大的学习率更稳定地进行梯度传播，甚至增加网络的泛化能力。

作者|言有三

1 什么是归一化/标准化

Normalization 是一个统计学中的概念，我们可以叫它归一化或者规范化，它并不是一个完全定义好的数学操作(如加减乘除)。它通过将数据进行偏移和尺度缩放调整，在数据预处理时是非常常见的操作，在网络的中间层如今也很频繁的被使用。

1.1 线性归一化

最简单来说，归一化是指将数据约束到固定的分布范围，比如 8 位图像的 0～255 像素值，比如 0～1。

在数字图像处理领域有一个很常见的线性对比度拉伸操作：

$$X=(x-xmin)/(xmax-mxin)$$

它常常可以实现下面的增强对比度的效果。

不过以上的归一化方法有个非常致命的缺陷，当 X 最大值或者最小值为孤立的极值点，会影响性能。

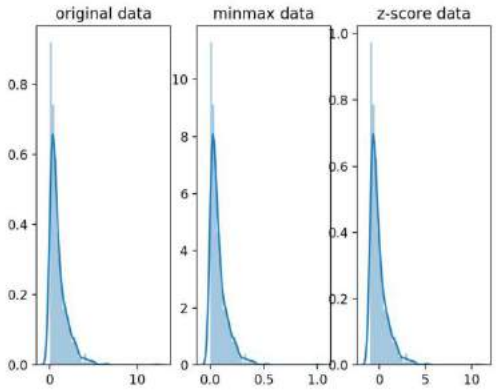
1.2. 零均值归一化/Z-score 标准化

零均值归一化也是一个常见的归一化方法，被称为标准化方法，即每一变量值与其平均值之差除以该变量的标准差。

$$y_i = \frac{x_i - \mu}{\sigma}$$

经过处理后的数据符合均值为 0，标准差为 1 的分布，**如果原始的分布是正态分布，那么 z-score 标准化就将原始的正态分布转换为标准正态分布**，机器学习中的很多问题都是基于正态分布的假设，这是更加常用的归一化方法。

以上两种方法都是线性变换，对输入向量 X 按比例压缩再进行平移，操作之后原始有量纲的变量变成无量纲的变量。不过它们不会改变分布本身的形状，下面以一个指数分布为例：



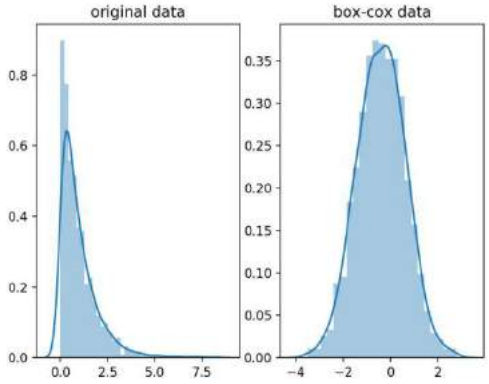
如果要改变分布本身的形状，下面也介绍两种。

1.3.正态分布 Box-Cox 变换

box-cox 变换可以将一个非正态分布转换为正态分布，使得分布具有对称性，变换公式如下：

$$Y^{(\lambda)} = \begin{cases} \dfrac{Y^{\lambda}-1}{\lambda}, & \lambda \neq 0 \\ ln\lambda, & \lambda = 0 \end{cases}$$

在这里 lamda 是一个基于数据求取的待定变换参数，Box-Cox 的效果如下。



1.4. 直方图均衡化

直方图均衡也可以将某一个分布归一化到另一个分布，它通过图像的灰度值分布，即图像直方图来对图像进行对比度进调整，可以增强局部的对比度。

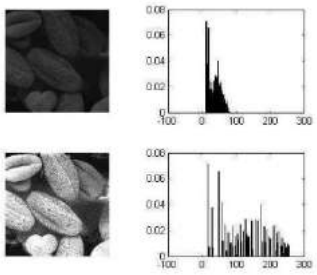
它的变换步骤如下：

(1)计算概率密度和累积概率密度。

(2)创建累积概率到灰度分布范围的单调线性映射 T。

(3)根据 T 进行原始灰度值到新灰度值的映射。

直方图均衡化将任意的灰度范围映射到全局灰度范围之间，对于 8 位的图像就是(0,255)，它相对于直接线性拉伸，让分布更加均匀，对于增强相近灰度的对比度很有效，如下图。



综上，归一化数据的目标，是为了让数据的分布变得更加符合期望，增强数据的表达能力。

在深度学习中，因为网络的层数非常多，如果数据分布在某一层开始有明显的偏移，随着网络的加深这一问题会加剧(这在 BN 的文章中被称之为 internal covariate shift)，进而导致模型优化的难度增加，甚至不能优化。所以，归一化就是要减缓这个问题。

2 Batch Normalization

2.1、基本原理

现在一般采用批梯度下降方法对深度学习进行优化，这种方法把数据分为若干组，按组来更新参数，一组中的数据共同决定了本次梯度的方向，下降时减少了随机性。另一方面因为批的样本数与整个数据集相比小了很多，计算量也下降了很多。

Batch Normalization(简称 BN)中的 batch 就是批量数据，即每一次优化时的样本数目，通常 BN 网络层用在卷积层后，用于重新调整数据分布。假设神经网络某层一个 batch 的输入为 X=[x1,x2,...,xn]，其中 xi 代表一个样本，n 为 batch size。

首先，我们需要求得 mini-batch 里元素的均值：

$$\mu_B = \frac{1}{n}\sum_{i=1}^{n} x_i$$

接下来，求取 mini-batch 的方差：

$$\sigma_B^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_B)^2$$

这样我们就可以对每个元素进行归一化。

$$x_i' = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

最后进行尺度缩放和偏移操作，这样可以变换回原始的分布，实现恒等变换，这样的目的是为了补偿网络的非线性表达能力，因为经过标准化之后，偏移量丢失。具体的表达如下，yi 就是网络的最终输出。

$$y_i = \gamma_i \cdot x_i' + \beta_i$$

假如 gamma 等于方差，beta 等于均值，就实现了恒等变换。

从某种意义上来说，gamma 和 beta 代表的其实是输入数据分布的方差和偏移。对于没有 BN 的网络，这两个值与前一层网络带来的非线性性质有关，而经过变换后，就跟前面一层无关，变成了当前层的一个学习参数，这更加有利于优化并且不会降低网络的能力。

对于 CNN，BN 的操作是在各个特征维度之间单独进行，也就是说各个通道是分别进行 Batch Normalization 操作的。

如果输出的 blob 大小为(N,C,H,W)，那么在每一层 normalization 就是基于 **N*H*W 个数值进行求平均以及方差的操作**，记住这里我们后面会进行比较。

2.2.BN 带来的好处。

(1) 减轻了对参数初始化的依赖，这是利于调参的朋友们的。

(2) 训练更快，可以使用更高的学习率。

(3) BN 一定程度上增加了泛化能力，dropout 等技术可以去掉。

2.3.BN 的缺陷

从上面可以看出，batch normalization 依赖于 batch 的大小，当 batch 值很小时，计算的均值和方差不稳定。研究表明对于 ResNet 类模型在 ImageNet 数据集上，batch 从 16 降低到 8 时开始有非常明显的性能下降，在训练过程中计算的均值和方差不准确，而在测试的时候使用的就是训练过程中保持下来的均值和方差。

这一个特性，导致 batch normalization 不适合以下的几种场景。

(1)batch 非常小，比如训练资源有限无法应用较大的 batch，也比如在线学习等使用单例进行模型参数更新的场景。

(2)rnn，因为它是一个动态的网络结构，同一个 batch 中训练实例有长有短，导致每一个时间步长必须维持各自的统计量，这使得 BN 并不能正确的使用。在 rnn 中，

对 bn 进行改进也非常的困难。不过，困难并不意味着没人做，事实上现在仍然可以使用的，不过这超出了咱们初识境的学习范围。

2.4.BN 的改进

针对 BN 依赖于 batch 的这个问题，BN 的作者亲自现身提供了改进，即在原来的基础上增加了一个仿射变换。

$$x_i' = \frac{x_i - \mu_\beta}{\sigma_\beta} \cdot \mathrm{r} + \mathrm{d}$$

其中参数 r，d 就是仿射变换参数，它们本身是通过如下的方式进行计算的

$$r = \frac{\sigma_\beta}{\sigma}, \quad \mathrm{d} = \frac{\mu_\beta - \mu}{\sigma}$$

其中参数都是通过滑动平均的方法进行更新

$$\mu := \mu + \alpha(\mu_\beta - \mu)$$
$$\sigma := \sigma + \alpha(\sigma_\beta - \sigma)$$

所以 r 和 d 就是一个跟样本有关的参数，通过这样的变换来进行学习，这两个参数在训练的时候并不参与训练。

在实际使用的时候，先使用 BN 进行训练得到一个相对稳定的移动平均，网络迭代的后期再使用刚才的方法，称为 Batch Renormalization，当然 r 和 d 的大小必须进行限制。

2.3 Batch Normalization 的变种

Normalization 思想非常简单，为深层网络的训练做出了很大贡献。因为有依赖于样本数目的缺陷，所以也被研究人员盯上进行改进。说的比较多的就是 Layer Normalization 与 Instance Normalization，Group Normalization 了。

前面说了 Batch Normalization 各个通道之间是独立进行计算，如果抛弃对 batch 的依赖，也就是每一个样本都单独进行 normalization，同时各个通道都要用到，就得到了 **Layer Normalization**。

跟 Batch Normalization 仅针对单个神经元不同，Layer Normalization 考虑了神经网络中一层的神经元。如果输出的 blob 大小为(N,C,H,W)，那么在每一层 Layer Normalization 就是基于 **C*H*W 个数值进行求平均以及方差的操作**。

Layer Normalization 把每一层的特征通道一起用于归一化，如果每一个特征层单独进行归一化呢？也就是限制在某一个特征通道内，那就是 **instance normalization 了**。

如果输出的 blob 大小为(N,C,H,W)，那么在每一层 Instance Normalization 就是基于 **H*W 个数值进行求平均以及方差的操作**。对于风格化类的图像应用，Instance Normalization 通常能取得更好的结果，它的使用本来就是风格迁移应用中提出。

Group Normalization 是 Layer Normalization 和 Instance Normalization 的中间体， Group Normalization 将 channel 方向分 group，然后对每个 Group 内做归一化，算其均值与方差。

如果输出的 blob 大小为(N,C,H,W)，将通道 C 分为 G 个组，那么 Group Normalization 就是基于 **G*H*W 个数值进行求平均以及方差的操作**。我只想说，你们真会玩，要榨干所有可能性。

在 Batch Normalization 之外，有人提出了通用版本 **Generalized Batch Normalization**，有人提出了硬件更加友好的 **L1-Norm Batch Normalization** 等，不再一一讲述。

另一方面，以上的 Batch Normalization，Layer Normalization，Instance Normalization 都是将规范化应用于输入数据 x，Weight normalization 则是对权重进行规范化，感兴趣的可以自行了解，使用比较少，也不在我们的讨论范围。

这么多的 Normalization 怎么使用呢？有一些基本的建议吧，不一定是正确答案。

(1) 正常的处理图片的 CNN 模型都应该使用 Batch Normalization。只要保证 batch size 较大(不低于 32)，并且打乱了输入样本的顺序。如果 batch 太小，则优先用 Group Normalization 替代。

(2)对于 RNN 等时序模型，有时候同一个 batch 内部的训练实例长度不一(不同长度的句子)，则不同的时态下需要保存不同的统计量，无法正确使用 BN 层，只能使用 Layer Normalization。

(3) 对于图像生成以及风格迁移类应用，使用 Instance Normalization 更加合适。

4 Batch Normalization 的思考

最后是关于 Batch Normalization 的思考，应该说，normalization 机制至今仍然是一个非常 open 的问题，相关的理论研究一直都有，大家最关心的是 Batch Normalization 怎么就有效了。

之所以只说 Batch Normalization，是因为上面的这些方法的差异主要在于计算 normalization 的元素集合不同。Batch Normalization 是 N*H*W，Layer

Normalization 是 C*H*W，Instance Normalization 是 H*W，Group Normalization 是 G*H*W。

关于 Normalization 的有效性，有以下几个主要观点：

(1) 主流观点，Batch Normalization 调整了数据的分布，不考虑激活函数，它让每一层的输出归一化到了均值为 0 方差为 1 的分布，这保证了梯度的有效性，目前大部分资料都这样解释，比如 BN 的原始论文认为的缓解了 Internal Covariate Shift(ICS)问题。

(2) 可以使用更大的学习率，文[2]指出 BN 有效是因为用上 BN 层之后可以使用更大的学习率，从而跳出不好的局部极值，增强泛化能力，在它们的研究中做了大量的实验来验证。

(3) 损失平面平滑。文[3]的研究提出，BN 有效的根本原因不在于调整了分布，因为即使是在 BN 层后模拟 ICS，也仍然可以取得好的结果。它们指出，BN 有效的根本原因是平滑了损失平面。之前我们说过，Z-score 标准化对于包括孤立点的分布可以进行更平滑的调整。

算了，让大佬先上吧。

[1] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[J]. arXiv preprint arXiv:1502.03167, 2015.

[2] Bjorck N, Gomes C P, Selman B, et al. Understanding batch normalization[C]//Advances in Neural Information Processing Systems. 2018: 7705-7716.

[3] Santurkar S, Tsipras D, Ilyas A, et al. How does batch normalization help optimization?[C]//Advances in Neural Information Processing Systems. 2018: 2488-2498.

5 总结

BN 层技术的出现确实让网络学习起来更加简单了，降低了调参的工作量，不过它本身的作用机制还在被广泛研究中。几乎就像是深度学习中没有 open 问题的一个缩影，BN 到底为何，还无定论，如果你有兴趣和时间，不妨也去踩一坑。

【AI 初识境】为了围剿 SGD 大家这些年想过的那十几招

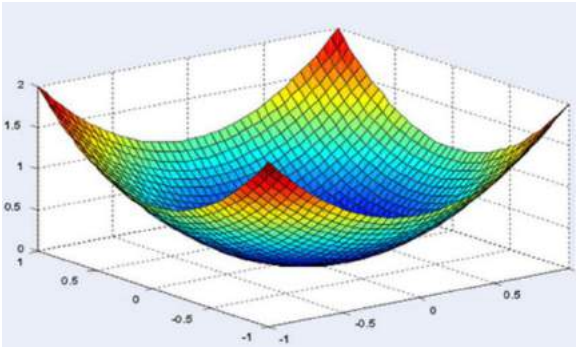
这是《AI 初识境》第 7 篇，这次我们说说常用的优化算法。所谓初识，就是对相关技术有基本了解，掌握了基本的使用方法。深度学习框架目前基本上都是使用一阶的梯度下降算法及其变种进行优化，在此基础上也发展出了很多的改进算法。另外，近年来二阶的优化算法也开始慢慢被研究起来。

作者|言有三

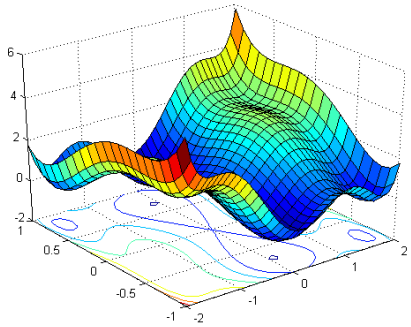
1 优化简述

深度学习模型的优化是一个非凸优化问题，这是与凸优化问题对应的。

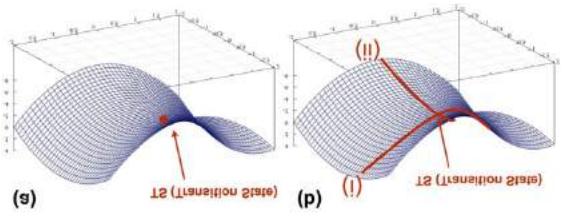
对于凸优化来说，任何局部最优解即为全局最优解。用贪婪算法或梯度下降法都能收敛到全局最优解，损失曲面如下。



而非凸优化问题则可能存在无数个局部最优点，损失曲面如下，可以看出有非常多的极值点，有极大值也有极小值。



除了极大极小值，还有一类值为“鞍点”，简单来说，它就是在某一些方向梯度下降，另一些方向梯度上升，形状似马鞍，如下图**红点就是鞍点**。



对于深度学习模型的优化来说，**鞍点比局部极大值点或者极小值点带来的问题更加严重**。

目前常用的优化方法分为一阶和二阶，这里的阶对应导数，一阶方法只需要一阶导数，二阶方法需要二阶导数。

常用的一阶算法就是：随机梯度下降 SGD 及其各类变种了。
常用的二阶算法就是：牛顿法等。

我们这里主要还是说一阶方法，二阶方法因为计算量的问题，现在还没有被广泛地使用。

2 梯度下降算法

本文目标不是为了从零开始讲清楚优化算法，所以有些细节和基础就略过。梯度下降算法，即通过梯度的反方向来进行优化，**批量梯度下降（Batch gradient descent）**用公式表述如下：

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

写成伪代码如下：

```
for i in range(nb_epochs):
        params_grad = evaluate_gradient(loss_function, data, params)
        params = params - learning_rate * params_grad
```

上面的梯度下降算法用到了数据集所有的数据，这在解决实际问题时通常是不可能，想想 Imagenet1000 有 100G 以上的图像，内存装不下，速度也很慢。

我们需要在线能够实时计算，于是一次取一个样本，就有了**随机梯度下降（Stochastic gradient descent），简称 sgd**。

公式如下：

$$\theta = \theta - \eta \cdot \Delta^{\theta} J(\theta; x_{(i)}; y_{(i)})$$

写成伪代码如下：

```
for i in range(nb_epochs):
      np.random.shuffle(data)
      for example in data:

                params_grad = evaluate_gradient(loss_function , example ,
params)
                params = params - learning_rate * params_grad
```

sgd 方法缺点很明显，梯度震荡，所以就有了后来大家常用的**小批量梯度下降算法（Mini-batch gradient descent）**。

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

伪代码如下：

```
for i in range(nb_epochs):
       np.random.shuffle(data)
       for batch in get_batches(data, batch_size=50):

                params_grad = evaluate_gradient(loss_function, batch, params)
                params = params - learning_rate * params_grad
```

下面我们要形成共识，**说 sgd 算法，实际上指的就是 mini-batch gradient descent 算法，没有人会去一次拿整个数据集或者一个样本进行优化。**

当然还是要总结一下 SGD 算法的毛病。

(1) 学习率大小和策略选择困难，想必动手经验丰富的自然懂。
(2) 学习率不够智能，对参数的各个维度一视同仁。
(3) 同时面临局部极值和鞍点的问题。

3 梯度下降算法改进

3.1 Momentum 动量法

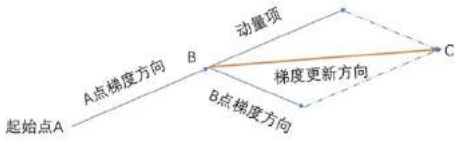
在所有的改进算法中，我觉得真正最有用的就是它。

前面说了梯度下降算法是按照梯度的反方向进行参数更新，但是**刚开始的时候梯度不稳定呀，方向改变是很正常的**，梯度就是抽疯了似的一下正一下反，导致做了很多无用的迭代。

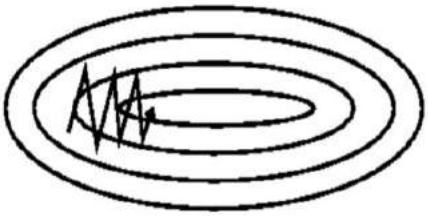
而动量法做的很简单，**相信之前的梯度**。如果梯度方向不变，就越发更新的快，反之减弱当前梯度。

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

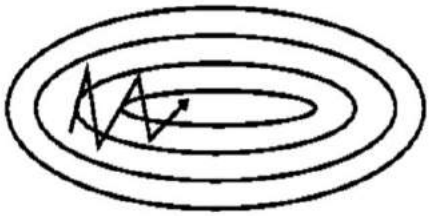
画成图就是这样。



效果对比就这意思。



(a) SGD without momentum



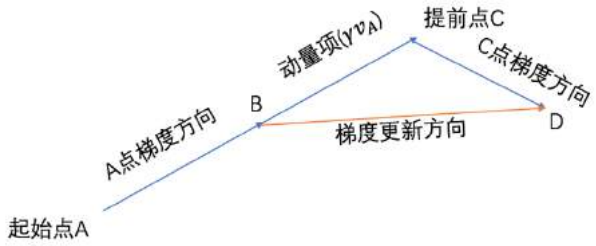
(b) SGD with momentum

3.2 Nesterov accelerated gradient 法 ，简称 NAG 算法

仍然是动量法，只是它要求这个下降更加智能。

既然动量法已经把前一次的梯度和当前梯度融合，那何不更进一步，直接先按照前一次梯度方向更新一步将它作为当前的梯度，看下面的式子就明白了。

$$v_t = \gamma\, v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$



如上图，自己领会。nesterov 的好处就是，当梯度方向快要改变的时候，它提前获得了该信息，从而减弱了这个过程，再次减少了无用的迭代。

3.3 Adagrad 法

思路很简单，**不同的参数是需要不同的学习率的**，有的要慢慢学，有的要快快学，所以就给了一个权重咯，而且是用了历史上所有的梯度幅值。

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$

3.4 Adadelta 与 Rmsprop

Adagrad 用了所有的梯度，问题也就来了，累加的梯度幅值是越来越大的。导致学习率前面的乘因子越来越小，后来就学不动了呀。

Adadelta 就只是动了一丢丢小心思，只累加了一个窗口的梯度，而且计算方法也更有效。

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

并且，将学习率用前一时刻参数的平方根来代替，最终更新算法变成了这样。

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

把 γ 变成 0.9，就是 RMSprop 方法了，这个方法在 Hinton 的课程中使用的，没有发表成论文，毕竟有 Adadelta 了，没有发表必要。与 adadelta 另一个不同是还是需要学习率的，建议 0.001。

3.5 Adam 方法

Adam 算法可能是除了 SGD 算法之外大家最熟悉的了，无脑使用，不需调参。Adam 对梯度的一阶和二阶都进行了估计与偏差修正，使用梯度的一阶矩估计和二阶矩估计来动态调整每个参数的学习率。

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

看出来了吧，与 Adadelta 和 Rmsprop 如出一辙，与 Momentum SGD 也颇为相似。上面的式子根据梯度对参数更新的幅度进行了动态调整，所以 Adam 对学习率没有那么敏感。

Adam 每次迭代参数的学习步长都有一个确定的范围，不会因为很大的梯度导致很大的学习步长，参数的值比较稳定，但是它也并非真的是参数不敏感的，学习率在训练的后期可仍然可能不稳定导致无法收敛到足够好的值，泛化能力较差，这在文[3]中有非常详细的研究，后面也会简单说一下。

3.6 AdaMax

将 Adam 使用的二阶矩变成更高阶，就成了 Adamax 算法。

$$\begin{aligned} u_t &= \beta_2^\infty v_{t-1} + (1-\beta_2^\infty)|g_t|^\infty \\ &= \max(\beta_2 \cdot v_{t-1}, |g_t|) \end{aligned}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t} \hat{m}_t$$

3.7 Nadam 法

Nag 加上 Adam，就成了 Nadam 方法，即带有动量项的 Adam，所以形式也很简单，如下，可以将其分别与 Adam 算法和 NAG 算法的式子比较看看。

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} (\beta_1 \hat{m}_t + \frac{(1-\beta_1) g_t}{1-\beta_1^t})$$

3.8 AMSgrad 方法

ICLR 2018 最佳论文提出了 AMSgrad 方法，研究人员观察到 Adam 类的方法之所以会不能收敛到好的结果，是因为在优化算法中广泛使用的指数衰减方法会使得梯度的记忆时间太短。

在深度学习中，每一个 mini-batch 对结果的优化贡献是不一样的，有的产生的梯度特别有效，但是也一视同仁地被时间所遗忘。

具体的做法是使用过去平方梯度的最大值来更新参数，而不是指数平均。

$$
\begin{aligned}
g_t &= \nabla f_t(x_t) \\
m_t &= \beta_{1t} m_{t-1} + (1-\beta_{1t}) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{v}_t &= \max(\hat{v}_{t-1}, v_t) \text{ and } \hat{V}_t = \mathrm{diag}(\hat{v}_t) \\
x_{t+1} &= \Pi_{\mathcal{F}, \sqrt{\hat{V}_t}}(x_t - \alpha_t m_t / \sqrt{\hat{v}_t})
\end{aligned}
$$

3.9 Adafactor 方法

Adam 算法有两个参数，beta1 和 beta2，相关研究表明 beta2 的值对收敛结果有影响，如果较低，衰减太大容易不收敛，反之就容易收敛不稳定。Adafactor 是通过给 beta1 和 beta2 本身也增加了一个衰减。

$$
\begin{aligned}
\hat{\beta}_{1t} &= \beta_1 \frac{1-\beta_1^{t-1}}{1-\beta_1^t} \\
\hat{\beta}_{2t} &= \beta_2 \frac{1-\beta_2^{t-1}}{1-\beta_2^t}
\end{aligned}
$$

beta2 的值刚开始是 0，之后随着时间的增加而逼近预设值。

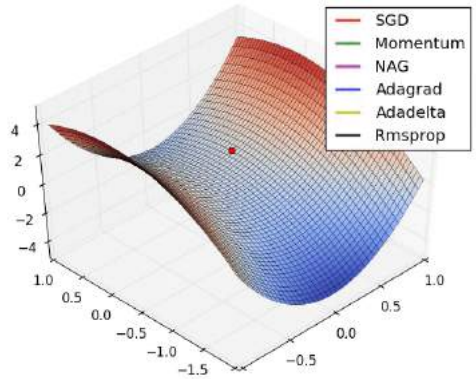
3.10 Adabound 方法

上面说了，beta2 的值造成 Adam 算法有可能不收敛或者不稳定而找不到全局最优解，落实到最后的优化参数那就是不稳定和异常(过大或者过小)的学习率。Adabound 采用的解决问题的方式就非常的简单了，那就是限制最大和最小值范围，约束住学习率的大小。

$$
\mathrm{Clip}(\alpha/\sqrt{V_t}, \eta_l, \eta_u),
$$

ηl(t)和ηu(t)分别是一个随着时间单调递增和递减的函数，最后两者收敛到同一个值。

说了这么多，对上面各种方法从一个鞍点开始优化，表现如何的预期效果图如下，参考文[1]。

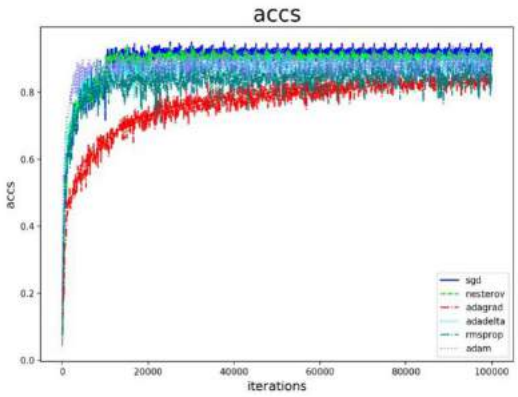


理论上，就是上面这样的。文章作者会告诉你对于数据稀疏的问题，用自适应学习率算法就好了，而且使用人家推荐的参数就好。其中，Adam 会最佳。

4 总结

4.1 改进方法是否都比 SGD 算法强？

上面说了这么多理论，分析起来头头是道，各种改进版本似乎各个碾压 SGD 算法。但是否真的如此。笔者曾经做过一个简单的实验，结果如下。



所有方法都采用作者们的默认配置，并且进行了参数调优，不好的结果就不拿出来了。

- nesterov 方法，与 sgd 算法同样的配置。
- adam 算法，m1=0.9，m2=0.999，lr=0.001。
- rms 算法，rms_decay=0.9，lr=0.001。
- adagrad，adadelta 学习率不敏感。

看起来好像都不如 SGD 算法，实际上这是一个很普遍的现象，各类开源项目和论文[3-4]都能够印证这个结论。

总体上来说，改进方法降低了调参工作量，只要能够达到与精细调参的 SGD 相当的性能，就很有意义了，这也是 Adam 流行的原因。但是，改进策略带来的学习率和步长的不稳定还是有可能影响算法的性能，因此这也是一个研究的方向，不然哪来这么多 Adam 的变种呢。

4.2 二阶方法研究的怎么样了呢？

二阶的方法因为使用了导数的二阶信息，因此其优化方向更加准确，速度也更快，这是它的优势。

但是它的劣势也极其明显，使用二阶方法通常需要直接计算或者近似估计 Hessian 矩阵，一阶方法一次迭代更新复杂度为 O(N)，二阶方法就是 O(N*N)，深层神经网络中变量实在是太多了，搞不动的。

不过，还是有研究者去研究的。比如东京工业大学和 NVIDIA 在[5]中使用的 K-FAC 方法，用 1024 块 Tesla V100 豪无人性地在 10 分钟内把 ImageNet 在 35 个 epoch 内训练到 75%的 top-1 精度。K-FAC 已经在 CNN 的训练中很常用了，感兴趣的可以去了解。

其他的二阶方法笔者也关注到了一些，以后等有了比较多稳定靠谱的研究，再来分享把。

[1] Ruder S. An overview of gradient descent optimization algorithms[J]. arXiv preprint arXiv:1609.04747, 2016.

[2] Reddi S J, Kale S, Kumar S. On the convergence of adam and beyond[J]. 2018.

[3] Bottou L, Curtis F E, Nocedal J. Optimization methods for large-scale machine learning[J]. Siam Review, 2018, 60(2): 223-311.

[4] Keskar N S, Socher R. Improving generalization performance by switching from adam to sgd[J]. arXiv preprint arXiv:1712.07628, 2017.

[5] Osawa K, Tsuji Y, Ueno Y, et al. Second-order Optimization Method for Large Mini-batch: Training ResNet-50 on ImageNet in 35 Epochs[J]. arXiv preprint arXiv:1811.12019, 2018.

如果想了解更多，欢迎关注知乎。

5 总结

其实大家不要紧张，一般做项目先上 Adam 试试，不行再换。等到了优化方法对项目

起关键作用的时候，说明你可能已经比较牛逼了，那还担心什么呢？

【AI 初识境】被 Hinton，DeepMind 和斯坦福嫌弃的池化，到底是什么？

这是专栏《AI 初识境》的第 8 篇文章。所谓初识，就是对相关技术有基本了解，掌握了基本的使用方法。本文来说说深度学习中的池化问题，包含池化的种类，作用机制以及最新的思考。

作者|言有三

1 池化还要不要了

这一次咱们反着来，说说学术界对池化的最新观点。通常我们认为，**池化可以增加网络对于平移的不变性，对于网络的泛化能力的提升是非常关键的**。不过，到底能起到多大的正向作用，却是被很多人怀疑的。

首先是 Hinton，还记得 Hinton 提出的 Capsule Module 吧。他认为池化的使用就是一个大错误，而它有效又反而是一个大灾难。池化固然可以提供一些平移和旋转不变性，但是也破坏了图像中的姿态和空间等信息，对检测分割等高级任务有影响，所以才提出胶囊网络(CapsuleNetwork)。至于这个发展的怎么样了，笔者没有关注，但是从大佬敢于革自己的“本命”这一点，就说明这个问题确实有点严重。

Hinton 虽然指出了 pooling 的坏影响，但是无法否定其好处，那么池化是不是真的能够提升网络的泛化能力呢？

首先站出来好好回答这个问题的是斯坦福大学 Eric Kauderer-Abrams 的研究【1】，它们通过一个平移敏感图来进行研究。

这个平移敏感图长下面这样，它评估的就是一个网络的输出对于输入的平移的敏感度。

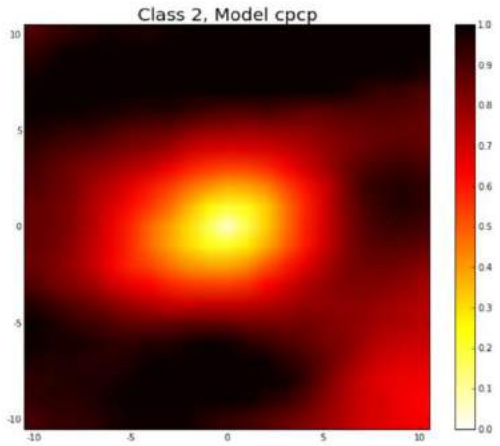
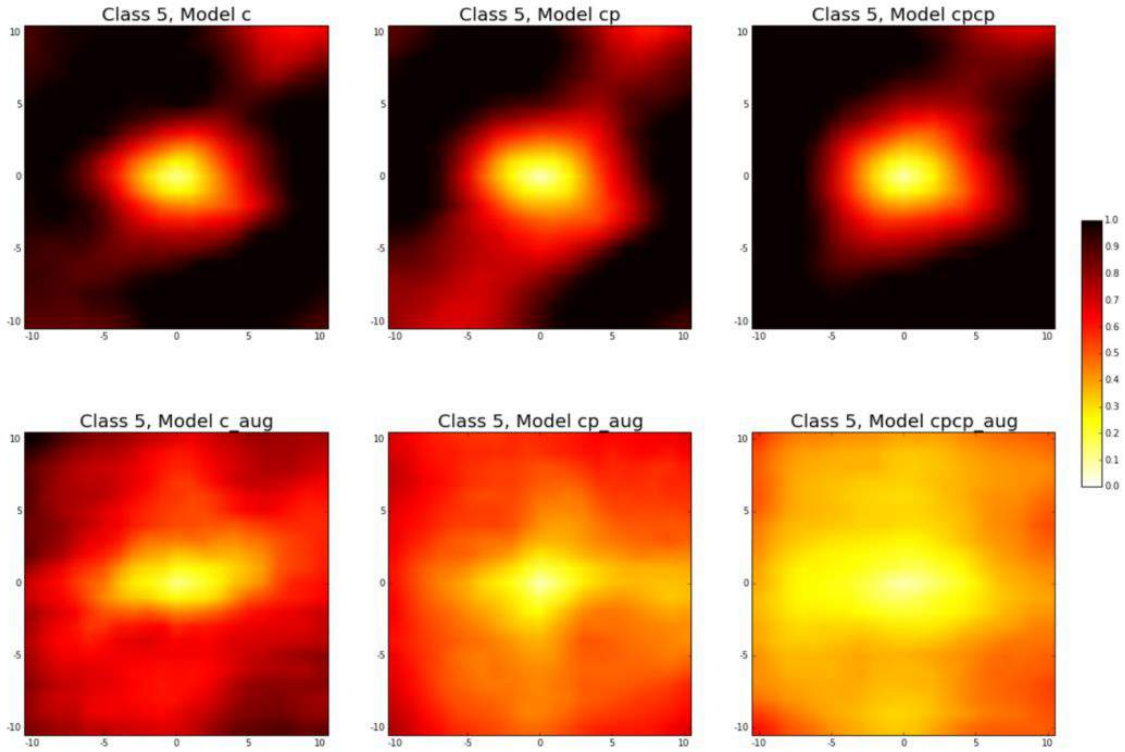


Figure 1. A translation-sensitivity map obtained from a network with two convolution and two pooling layers.

上面这个图是这么算的，首先用原图计算预测特征分数，然后用平移过的图计算预测特征分数，最后计算两者的归一化分数，越亮说明越相关。x 和 y 分别就是偏移量，可以看到 x，y 都接近 0 的时候越亮，说明越相关，然后就随着距离的增强而降低。

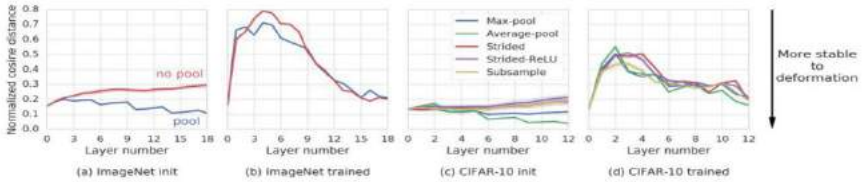
这就是说平移越大之后，对性能的影响越大，毕竟一个网络不可能拥有完全的平移不变性。

在这个基础上，他们就做实验了，结果如下，c 表示卷积，p 表示 pooling，aug 表示数据增强，所以这里就是比较 pooling 和 aug 对性能的影响，结果表明**池化不池化的，好像没有什么用，而数据增强做不做得好，才是关键。结果说明 CNN 本身没什么平移不变性，是靠数据学来的。**



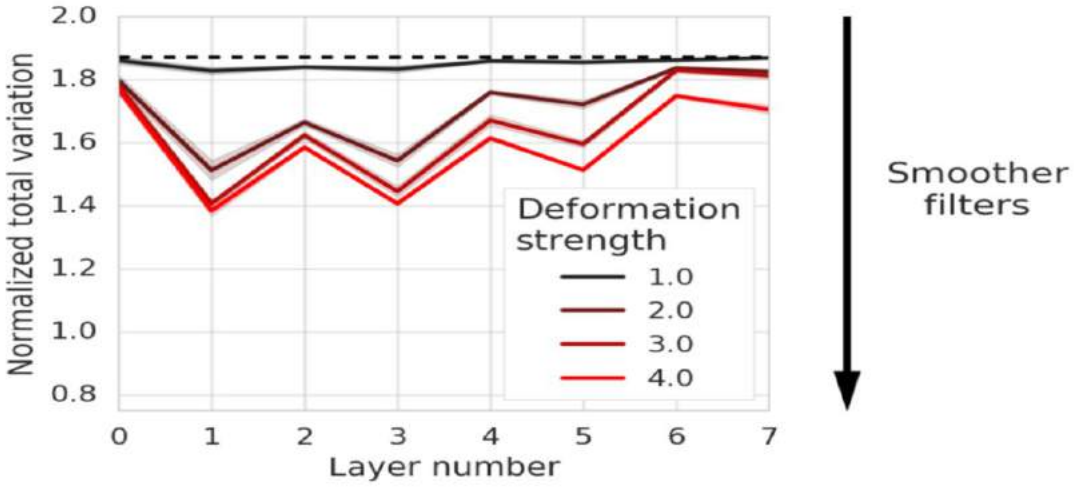
上面做了实验，但是没有更深层次地分析，为什么池化就没有用了呢，这可是违反我们的常识的。

DeepMind 的研究【2】给出了一个比较有说服力的解答，实验的设置差不多，使用非池化和各种池化的网络结构。



总之结论就是：看上面的 4 个图。**(a) 刚开始的时候池化确实有利于提高抗变形能力。(b) 不管池化不池化，模型最后学习完都能获得同样的抗变形能力。(c) 初始化的时候不同的池化方法是有差异的。(d) 学习完之后不管什么池化方法效果都差不多。**

那总得有个理由吧？他们给出的理由是卷积核本身参数越平滑才越能提高对平移的稳定性，文中在卷积操作后面串接平滑操作，实验对比如下。



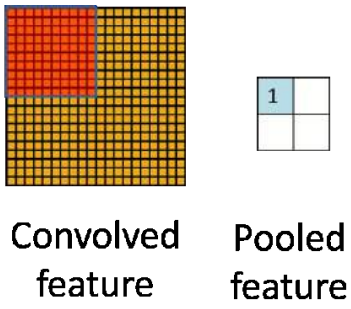
这也没毛病对吧，卷积核都平滑了，当然就没有那么敏感了。

暂且先总结一下吧：池化什么的不重要了，搞数据增强才是正道。

2 什么是池化

上面都这么说了，接下来说池化略有点尴尬，但是作为知识体系的重要一环，还是有必要讲述。

pooling，小名池化，思想来自于视觉机制，是对信息进行抽象的过程。



上图就是一个池化的示意图，用了一个 10*10 的卷积核，对 20*20 的图像分块不重叠的进行了池化，池化之后 featuremap 为 2*2 的大小。

pooling 有什么用呢？或者说为什么需要 pooling 呢？原因有几个：

1、增大感受野

所谓感受野，即一个像素对应回原图的区域大小，假如没有 pooling，一个 3*3，步长为 1 的卷积，那么输出的一个像素的感受野就是 3*3 的区域，再加一个 stride=1 的 3*3 卷积，则感受野为 5*5。

假如我们在每一个卷积中间加上 3*3 的 pooling 呢？很明显感受野迅速增大，这就是 pooling 的一大用处。感受野的增加对于模型的能力的提升是必要的，正所谓“一叶障目则不见泰山也”。

2、平移不变性

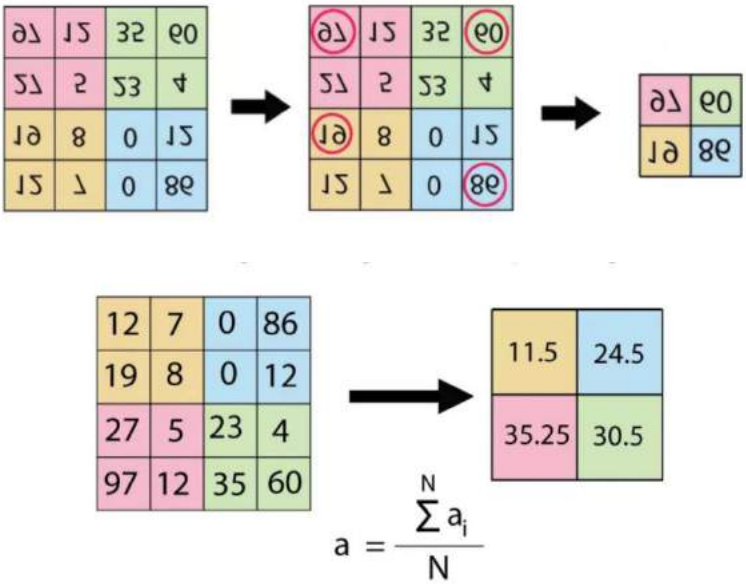
我们希望目标的些许位置的移动，能得到相同的结果。因为 pooling 不断地抽象了区域的特征而不关心位置，所以 pooling 一定程度上增加了平移不变性。

3、降低优化难度和参数

我们可以用步长大于 1 的卷积来替代池化，但是池化每个特征通道单独做降采样，与基于卷积的降采样相比，不需要参数，更容易优化。全局池化更是可以大大降低模型的参数量和优化工作量。

3 池化有哪些

1、平均池化和最大池化



这是我们最熟悉的，通常认为如果选取区域均值(mean pooling)，往往能保留整体数据的特征，较好的突出背景信息；如果选取区域最大值(max pooling)，则能更好保留纹理特征。

2、stochastic pooling/mixed pooling

stochastic pooling 对 feature map 中的元素按照其概率值大小随机选择，元素被选中的概率与其数值大小正相关，这就是一种正则化的操作了。mixed pooling 就是在 max/average pooling 中进行随机选择。

3、Data Driven/Detail-Preserving Pooling

上面的这些方法都是手动设计，而现在深度学习各个领域其实都是往自动化的方向发展。

我们前面也说过，从激活函数到归一化都开始研究数据驱动的方案，池化也是如此，每一张图片都可以学习到最适合自己的池化方式。

此外还有一些变种如 weighted max pooling，Lp pooling，generalization max pooling 就不再提了，还有 global pooling。

4 总结

带步长的卷积虽然不需要池化，却没有了灵活的激活机制。平均池化稳扎稳打，却丢失了细节。最大池化克服了平均池化的缺点，却打断了梯度回传。

最终发现，池化也还是要学的好，所谓随机应变，盖莫如此。另外，如何选择好用于池化的区域，也是一门学问。

参考文献

[1] Kaudererabrams E. Quantifying Translation-Invariance in Convolutional Neural Networks.[J]. arXiv: Computer Vision and Pattern Recognition, 2018.

[2] Ruderman A, Rabinowitz N C, Morcos A S, et al. Pooling is neither necessary nor sufficient for appropriate deformation stability in CNNs[J]. arXiv: Computer Vision and Pattern Recognition, 2018.

【AI 初识境】如何增加深度学习模型的泛化能力

这是专栏《AI 初识境》的第 9 篇文章。所谓初识，就是对相关技术有基本了解，掌握了基本的使用方法。本文来说说深度学习中的 generalization 问题，也就是泛化和正则化有关的内容。

作者|言有三

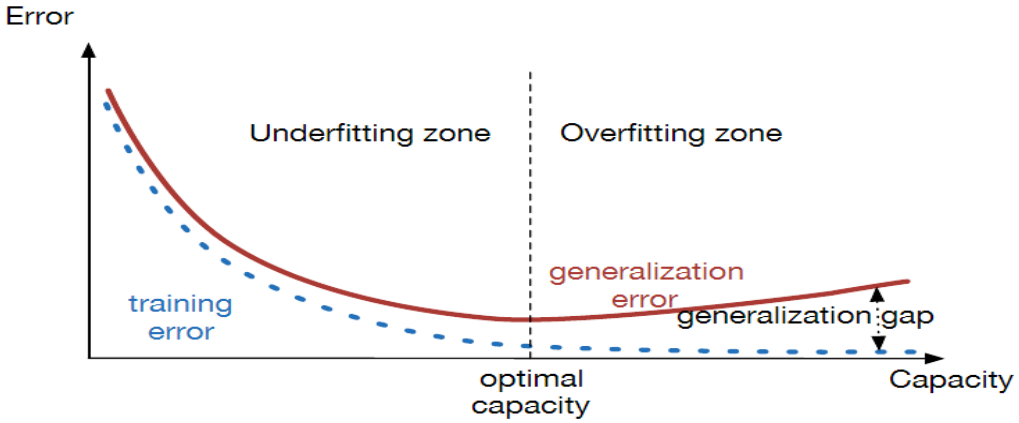
1 什么是 generalization

机器学习方法训练出来一个模型，希望它不仅仅是对于已知的数据(训练集)性能表现良好，对于未知的数据(测试集)也应该表现良好，也就是具有良好的 **generalization 能力**，这就是**泛化能力。测试集的误差，也被称为泛化误差**。

举个例子来说，我们在 ImageNet 上面训练分类模型，希望这个模型也能正确地分类我们自己拍摄的照片。

在机器学习中，泛化能力的好坏，最直观表现出来的就是模型的**过拟合(overfitting)与欠拟合(underfitting)。**

过拟合和欠拟合是用于描述模型在训练过程中的两种状态，一般来说，训练会是这样的一个曲线。下面的 training error，generalization error 分别是训练集和测试集的误差。



训练刚开始的时候，模型还在学习过程中，训练集和测试集的性能都比较差，这个时候，模型还没有学习到知识，处于欠拟合状态，曲线落在 **underfitting zone**，随着训练的进行，训练误差和测试误差都下降。

随着模型的进一步训练，在训练集上表现的越来越好，终于在突破一个点之后，训练集的误差下降，测试集的误差上升了，这个时候就进入了过拟合区间 **overfitting zone**。

不过也不是说什么训练过程，都会满足上面的曲线。

(1) 模型训练过程中，训练集的误差一定一直低于测试集吗？未必。

如果这两个集合本来就取自于同样的数据分布，比如从一个数据集中随机采样，那么有可能测试的误差从一开始就低于训练集。不过，总体的趋势肯定是不变的，两者从一开始慢慢下降直到最后过拟合，训练集的误差低于测试集。

(2) 模型的训练一定会过拟合吗？这也不一定！

如果数据集足够大，很可能模型的能力不够始终都不会过拟合。另一方面，有很多的方法可以阻止，或者减缓模型的过拟合，比如正则化，这就是下面第二部分要说的。

2 什么是 Regularization

Regularization 即正则化，它本是代数几何中的一个概念，我们不说因为说不好。放到机器学习里面来说，所谓正则化，它的目标就是要同时让**经验风险和模型复杂度较小**。

$$\min_f \sum_{i=1}^{n} V(f(x_i), y_i) + \lambda R(f)$$

以上是我们的优化目标，V 就是损失函数，它表示的是当输入 xi 预测输出为 f(xi)，而真实标签为 yi 时，应该给出多大的损失。那么我们不禁要问，有这一项不就行了吗？为什么还需要后面的那一项呢？R(f)又是什么呢？

这就是回到上面的泛化误差和过拟合的问题了，一个机器学习系统，学习的是从输入到输出的关系，**只要一个模型足够复杂，它是不是可以记住所有的训练集合样本之间的映射，代价就是模型复杂，带来的副作用就是没见过的只是略有不同的样本可能表现地就很差，就像下面这张图，只是更改了一个像素，预测就从 Dog 变成了 Cat。**

造成这种情况的问题就是学的太过，参数拟合的太好以致于超过了前面那个训练曲线的最低泛化误差临界点，究其根本原因是模型的表达能力足够强大到过拟合数据集。

式子中的 R(f)，正是为了约束模型的表达能力，f 是模型，R 是一个跟模型复杂度相关的函数，单调递增。

有同学可能会说，模型的表达能力跟模型大小，也就是参数量有关，限制模型的表达能力不是应该去调整模型大小吗？这里咱们从另一个角度来看模型的能力问题。

如果我们限制一层神经网络的参数只能为 0 或者 1，它的表达能力肯定不如不做限制，所以同样的参数量，模型的表达能力与参数本身也有关系，正则项就可以在参数上做文章。

所以说正则化就用于提高模型的泛化能力，这里所说的仅仅是狭义上的参数正则化，而广义上的正则化方法众多，第 3 部分进行介绍。

正则化的最终目标用一句土话来说，就是让网络学的不要太死，否则变成僵硬的书呆子。

3 正则化方法有哪些

正则化方法，根据具体的使用策略不同，有直接提供正则化约束的**参数正则化方法如 L1/L2 正则化**，以及通过工程上的技巧来实现更低泛化误差的方法，比如训练提前终止和模型集成，我将其称为**经验正则化**，也有**不直接提供约束的隐式正则化方法**如数据增强等，下面就从这三类进行讲述。

1、经验正则化方法

这里主要包含两种方法，即**提前终止和模型集成**。

(1) 提前终止

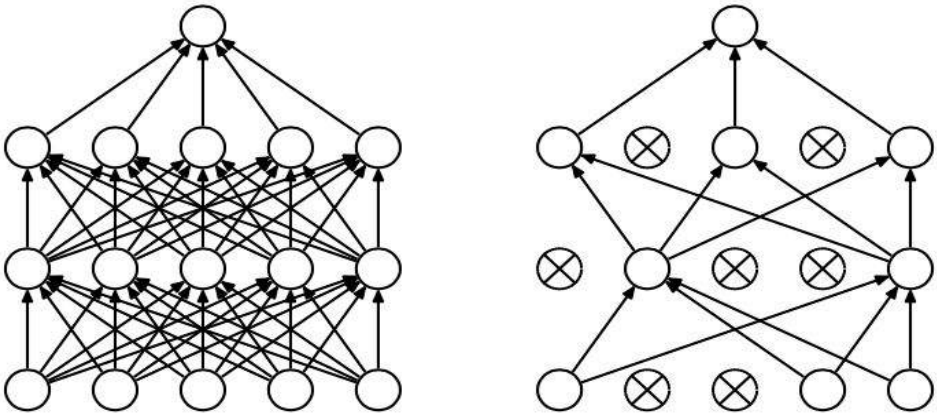
前面我们看的训练曲线随着不断迭代训练误差不断减少，但是泛化误差减少后开始增长。假如我们在泛化误差指标不再提升后，提前结束训练，也是一种正则化方法，这大概是最简单的方法了。

(2) 模型集成

另一种方法就是模型集成(ensemable)，也就是通过训练多个模型来完成该任务，它可以是不同网络结构，不同的初始化方法，不同的数据集训练的模型，也可以是用不同的测试图片处理方法，总之，采用多个模型进行投票的策略。

在这一类方法中，有一个非常有名的方法，即 Dropout。

Dropout 在 2014 年被 H 提出后在深度学习模型的训练中被广泛使用。它在训练过程中，随机的丢弃一部分输入，此时丢弃部分对应的参数不会更新。所谓的丢弃，其实就是让激活函数的输出为 0。结构示意图如下。

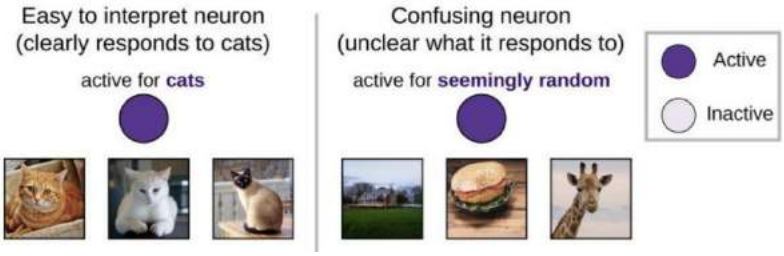


因而，对于一个有 n 个节点的神经网络，有了 dropout 后，就可以看做是 2^n 个模型的集合了，使用的时候当然不能用 2^n 个模型来进行推理，而是采用了近似方法，即在使用的时候不进行权重丢弃。根据丢弃比例的不同，在测试的时候会给输出乘以相应的系数，比如某一层在训练的时候只保留 50%的权重，在测试的时候是需要用到所有参数的，这个时候就给该层的权重乘以 0.5。

关于 dropout 的有效性，从结构上来说，它消除或者减弱了神经元节点间的联合，降低了网络对单个神经元的依赖，从而增强了泛化能力。不过也有另外的一些研究从数据增强的角度来思考这个问题。

那么，就真的不担心 dropout 会把一些非常重要的神经元删除吗？最新的神经科学的研究以及 DeepMind 等研究人员通过对神经元进行随机删除来研究网络性能，发现虽然某些神经元确实很重要，它们会选择性激活特定输入，比如只对输入猫图特别敏感，对其他输入则完全不感冒，但是删除这一类神经元仍然不影响网络能识别到猫。

这说明网络中未必少了谁就不行。不过反过来，上面说到的单个像素的攻击，则说明又有某些神经元至关重要。关于这其中的关系，仍然是研究热门，还是不断跟进更多最新的研究吧。



总之一句话，不怕删了谁。就 dropout 的使用方法而言，我们平常只更改 dropout 的比例，作者对此还有更多的建议。

(1) 因为 dropout 降低了模型的性能，所以对于原本需要容量为 N 的网络才能解决的问题，现在需要 N/p，p 就是保留该节点的概率，这个概率通常在 0.5～0.9 之间，p=1 就是普通的网络了。

(2) 因为 dropout 相当于增加了噪声，造成梯度的损失，所以需要使用更大的学习率和动量项。与此同时，对权重进行 max-norm 等权重约束方法，使其不超过某个值。

(3) 训练更久，很好理解。

对 dropout 方法，还有很多的变种，包括 **dropout connect，maxout，stochastic depth** 等。

一个神经元的输出实际上是由输入以及参数来共同决定，dropout 把神经元的值设置为 0 了，那是不是也可以把参数设置为 0 呢？这就是 drop connect，而且它可以比 dropout 更加灵活，可视为 Dropout 的一般化形式，从模型集成的角度来看，Dropout 是 2^n 个模型的平均，那 DropConnect 呢？它应该更多，因为权重连接的数目比节点数本身更多，所以 DropConnect 模型平均能力更强。

Drop Connect 和 Dropout 均引入了稀疏性，不同之处在于 Drop Connect 引入的是权重的稀疏而不是层的输出向量的稀疏。

另外，在 dropout 这一个思路上做相关文章的还有一些，比如 maxout，是一种激活函数，它对 N 个输入选择最大的作为激活输出。比如随机 pooling，是一种池化方法。比如 stochastic depth，它用在带有残差结构的网络中，将某些 res block 直接设置为等价映射。还有 backdrop，在前向的时候不 drop，在梯度反传的时候才做。

在这里不禁想给大家提两个问题

(1) 你还能想到多少种 drop 方法？想不到就等我们下次专门说。

(2) dropout 应该怎么跟其他的方法结合，比如 batch normalization，会强强联合得到更好的结果吗？

2、参数正则化方法

L2/L1 正则化方法，就是最常用的正则化方法，它直接来自于传统的机器学习。

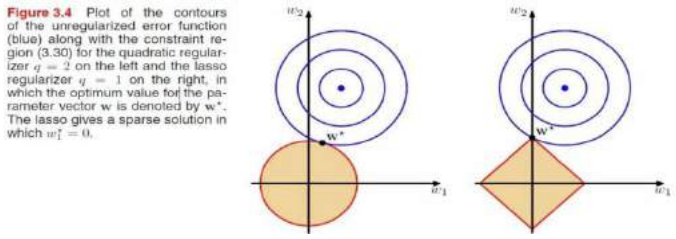
L2 正则化方法如下：

$$\tilde{J}(\omega; X, y) = J(\omega; X, y) + \frac{1}{2}\|\omega\|^2$$

L1 正则化方法如下：

$$\tilde{J}(\omega; X, y) = J(\omega; X, y) + \alpha\|\omega\|_1$$

那它们俩有什么区别呢？最流行的一种解释方法来自于模式识别和机器学习经典书籍，下面就是书中的图。



这么来看上面的那张图，参数空间(w1,w2)是一个二维平面，蓝色部分是一个平方损失函数，黄色部分是正则项。

蓝色的那个圈，中心的点其实代表的就是损失函数最优的点，而同心圆则代表不同的参数相同的损失，可见随着圆的扩大，损失增大。黄色的区域也类似，周边的红色线表示的是损失相同点的轮廓。

正则项的红色轮廓线示平方损失的蓝色轮廓线总要相交，才能使得两者加起来的损失最小，两者的所占区域的相对大小，是由权重因子决定的。不管怎么说，它们总有一个交叉点。

对于 L2 正则化，它的交点会使得 w1 或者 w2 的某一个维度特别小，而 L1 正则化则会使得 w1 或者 w2 的某一个维度等于 0，因此获得所谓的稀疏化。

在深度学习框架中，大家比起 L1 范数，更钟爱 L2 范数，因为它更加平滑和稳定。

3、隐式正则化方法

前面说了两种正则化方法，第一种，通过对网络结构的修改或者在使用方法上进行调整。第二种，直接对损失函数做了修改。这两种方法，其实都应该算作**显式的正则化方法**，因为在做这件事的过程中， 我们是有意识地知道自己在做正则化。

但是还有另一种正则化方法，它是**隐式的正则化方法**，并非有意识地直接去做正则化，却甚至能够取得更好的效果，这便是数据有关的操作，包括归一化方法和数据增强，扰乱标签。

关于数据增强的方法，我们以前已经有过几期文章讲述，从有监督方法到无监督方法。关于归一化方法，以 batch normalization 为代表，我们以前也详细地讲过。参考链接就放这里了，大家自取。

【AI 初识境】深度学习模型中的 Normalization，你懂了多少？
【技术综述】深度学习中的数据增强（下）
[综述类] 一文道尽深度学习中的数据增强方法（上）

实验表明，隐式的方法比显式的方法更强，从 batch normalization 的使用替换掉了 dropout，以及数据扩增碾压一切 trick 就可以看出。另外，批量随机梯度算法本身，也可以算是一种隐式的正则化方法，它随机选择批量样本而不是整个数据集，与上面的 dropout 方法其实也有异曲同工之妙。

这么看来，其实 data dependent 方法更好，咱们前面的几期都说过，不要闷着头设计，从数据中学习才是王道。

4 深度学习泛化能力到底好不好

你说深度学习的泛化能力是强还是不强，感觉完全可以打一架。

一方面，深度学习方法已经在各行各业落地，说泛化能力不好谁都不信，都已经经得起工业界的考验。关于如何定量的衡量泛化能力，目前从模型复杂度的角度有一些指标，可以参考[1]。

但是另一方面，有许多的研究[2-3]都表明，仅仅是对图像作出小的改动，甚至是一个像素的改动，都会导致那些强大的网络性能的急剧下降，这种不靠谱又让人心慌，在实际应用的过程中，笔者也一直遇到这样的问题，比如下图微小的平移操作对输出概率的严重影响，真的挺常见。

正则化方法可以完美解决吗？甚至最强的数据增强方法能做的都是有限的，一个网络可以记忆住样本和它的随机标签[4]，做什么正则化都没有作用。说起来神经网络一直在力求增强各种不变性，但是却往往搞不定偏移，尺度缩放。这就是为什么在刷比赛的时候，仅仅是对图像采用不同的 crop 策略，就能胜过任何其他方法的原因，从这一点来说，是一件非常没有意思的事情。

或许，关于正则化，再等等吧。

参考文献

[1] Neyshabur B, Bhojanapalli S, Mcallester D A, et al. Exploring Generalization in Deep Learning[J]. neural information processing systems, 2017: 5947-5956.

[2] Su J, Vargas D V, Sakurai K, et al. One Pixel Attack for Fooling Deep Neural Networks[J]. IEEE Transactions on Evolutionary Computation, 2019: 1-1.

[3] Azulay A, Weiss Y. Why do deep convolutional networks generalize so poorly to small image transformations[J]. arXiv: Computer Vision and Pattern Recognition, 2018.

[4] Zhang C, Bengio S, Hardt M, et al. Understanding deep learning requires rethinking generalization[J]. international conference on learning representations, 2017.

5 总结

深度学习基础理论部分就到此结束了，我们讲述了激活机制，参数初始化，归一化机制，池化机制，最优化方法，以及正则化，这些都对模型的性能和训练有影响，希望大家能够掌握基础知识。在我们的下一个境界，将回过头来继续看它们，以及更复杂的东西。

【AI 初识境】深度学习模型评估，从图像分类到生成模型

这是《AI 初识境》第 10 篇，这次我们说说深度学习模型常用的评价指标。所谓初识，就是对相关技术有基本了解，掌握了基本的使用方法。

凡事用数据说话，一个深度学习模型在各类任务中的表现都需要定量的指标进行评估，才能够进行横向的 PK 比较，今天就来说说分类，回归，质量评估，生成模型中常用的指标，以计算机视觉任务为例。

作者|言有三

1 分类评测指标

图像分类是计算机视觉中最基础的一个任务，也是几乎所有的基准模型进行比较的任务，从最开始比较简单的 10 分类的灰度图像手写数字识别 mnist，到后来更大一点的 10 分类的 cifar10 和 100 分类的 cifar100，到后来的 imagenet，图像分类任务伴随着数据库的增长，一步一步提升到了今天的水平。现在在 Imagenet 这样的超过 1000 万图像，2 万类的数据集中，计算机的图像分类水准已经超过了人类。

图像分类，顾名思义就是一个模式分类问题，它的目标是将不同的图像，划分到不同的类别，实现最小的分类误差，这里我们只考虑单标签分类问题，即每一个图片都有唯一的类别。

对于单个标签分类的问题，评价指标主要有 Accuracy，Precision，Recall，F-score，PR 曲线，ROC 和 AUC。

在计算这些指标之前，我们先计算几个基本指标，这些指标是基于二分类的任务，也可以拓展到多分类。计标签为正样本，分类为正样本的数目为 True Positive，简称 TP。标签为正样本，分类为负样本的数目为 False Negative，简称 FN。标签为负样本，分类为正样本的数目为 False Positive，简称 FP。标签为负样本，分类为负样本的数目为 True Negative，简称 TN。

判别是否为正例只需要设一个概率阈值 T，预测概率大于阈值 T 的为正类，小于阈值 T 的为负类，默认就是 0.5。如果我们减小这个阀值 T，更多的样本会被识别为正类，这样可以提高正类的召回率，但同时也会带来更多的负类被错分为正类。如果增加阈值 T，则正类的召回率降低，精度增加。如果是多类，比如 ImageNet1000 分类比赛中的 1000 类，预测类别就是预测概率最大的那一类。

1.1 准确率 Accuracy

单标签分类任务中每一个样本都只有一个确定的类别，预测到该类别就是分类正确，没有预测到就是分类错误，因此最直观的指标就是 Accuracy，也就是准确率。

Accuracy=(TP+TN)/(TP+FP+TN+FN)，表示的就是所有样本都正确分类的概率，可以使用不同的阈值 T。

在 ImageNet 中使用的 Accuracy 指标包括 Top_1 Accuracy 和 Top_5 Accuracy，Top_1 Accuracy 就是前面计算的 Accuracy。

记样本 xi 的类别为 yi，类别种类为(0,1,…,C)，预测类别函数为 f，则 Top-1 的计算方法如下：

$$\mathbf{Top1\_Acc} = \frac{\sum_{i=0}^{N-1}(f(x_i) == y_i)}{N}$$

如果给出概率最大的 5 个预测类别，只要包含了真实的类别，则判定预测正确，计算出来的指标就是 Top-5。

目前在 ImageNet 上，Top-5 的指标已经超过 95%，而 Top-1 的指标还在 80%左右。

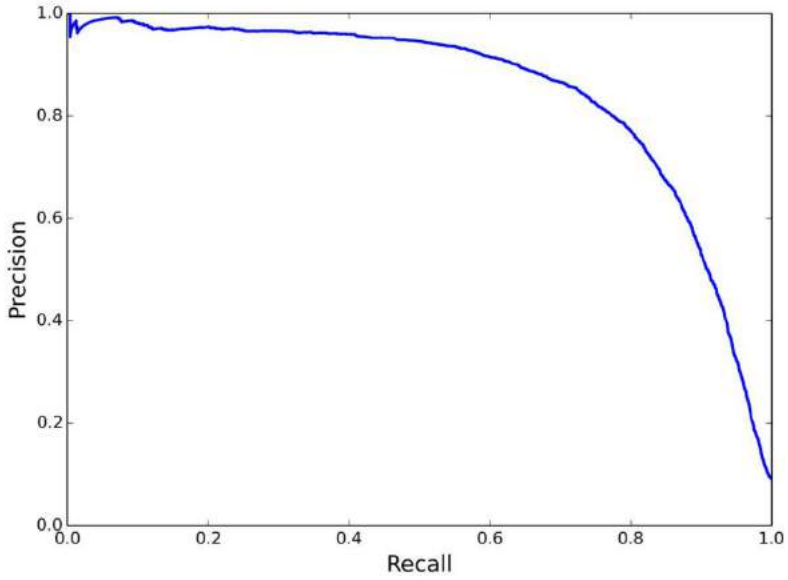
1.2.精确度 Precision 和召回率 Recall

如果我们只考虑正样本的指标，有两个很常用的指标，精确度和召回率。

正样本精确率为：Precision=TP/(TP+FP)，表示的是召回为正样本的样本中，到底有多少是真正的正样本。

正样本召回率为：Recall=TP/(TP+FN)，，表示的是有多少样本被召回类。当然，如果对负样本感兴趣的，也可以计算对应的精确率和召回率，这里记得区分精确率和准确率的分别。

通常召回率越高，精确度越低，根据不同的值可以绘制 Recall-Precision 曲线，如下。



横轴就是 recall，纵轴就是 precision，曲线越接近右上角，说明其性能越好，可以用该曲线与坐标轴包围的面积来定量评估，值在 0～1 之间。

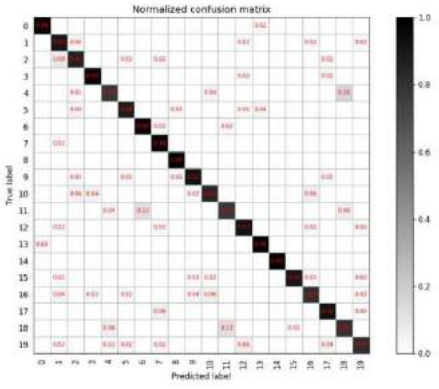
1.3. F1 score

有的时候我们不仅关注正样本的准确率，也关心其召回率，但是又不想用 Accuracy 来进行衡量，一个折中的指标是采用 F-score。

F1 score=2·Precision·Recall/(Precision+Recall)，只有在召回率 Recall 和精确率 Precision 都高的情况下，F1 score 才会很高，因此 F1 score 是一个综合性能的指标。

1.4. 混淆矩阵

如果对于每一类，我们想知道类别之间相互误分的情况，查看是否有特定的类别之间相互混淆，就可以用混淆矩阵画出分类的详细预测结果。对于包含多个类别的任务，混淆矩阵很清晰的反映出各类别之间的错分概率，如下。



这是一个包含 20 个类别的分类任务，混淆矩阵为 20*20 的矩阵，其中第 i 行第 j 列，表示第 i 类目标被分类为第 j 类的概率，可以知道，越好的分类器对角线上的值更大，其他地方应该越小。

1.5.ROC 曲线与 AUC 指标

以上的准确率 Accuracy，精确度 Precision，召回率 Recall，F1 score，混淆矩阵都只是一个单一的数值指标，如果我们想观察分类算法在不同的参数下的表现情况，就可以使用一条曲线，即 ROC 曲线，全称为 receiver operating characteristic。

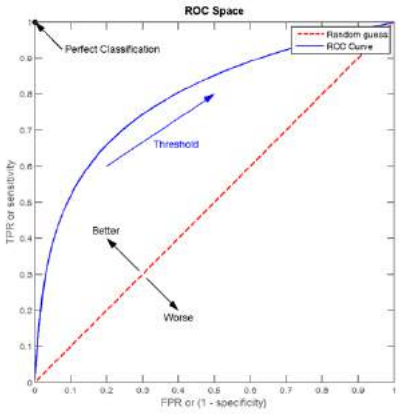
ROC 曲线可以用于评价一个分类器在不同阈值下的表现情况。

在 ROC 曲线中，每个点的横坐标是 false positive rate(FPR)，纵坐标是 true positive rate(TPR)，描绘了分类器在 True Positive 和 False Positive 间的平衡，两个指标的计算如下：

TPR=TP/(TP+FN)，代表分类器预测的正类中实际正实例占所有正实例的比例。

FPR=FP/(FP+TN)，代表分类器预测的正类中实际负实例占所有负实例的比例，FPR 越大，预测正类中实际负类越多。

ROC 曲线通常如下：



其中有 4 个关键的点：

点(0,0)：FPR=TPR=0，分类器预测所有的样本都为负样本。

点(1,1)：FPR=TPR=1，分类器预测所有的样本都为正样本。

点(0,1)：FPR=0, TPR=1，此时 FN＝0 且 FP＝0，所有的样本都正确分类。

点(1,0)：FPR=1，TPR=0，此时 TP＝0 且 TN＝0，最差分类器，避开了所有正确答案。

ROC 曲线相对于 PR 曲线有个很好的特性：当测试集中的正负样本的分布变化的时候，ROC 曲线能够保持不变，即对正负样本不均衡问题不敏感。

比如负样本的数量增加到原来的 10 倍，TPR 不受影响，FPR 的各项也是成比例的增加，并不会有太大的变化。所以不均衡样本问题通常选用 ROC 作为评价标准。

ROC 曲线越接近左上角，该分类器的性能越好，若一个分类器的 ROC 曲线完全包住另一个分类器，那么可以判断前者的性能更好。

如果我们想通过两条 ROC 曲线来定量评估两个分类器的性能，就可以使用 **AUC 指标**。

AUC（Area Under Curve）为 ROC 曲线下的面积，它表示的就是一个概率，这个面积的数值不会大于 1。随机挑选一个正样本以及一个负样本，AUC 表征的就是有多大的概率，分类器会对正样本给出的预测值高于负样本，当然前提是正样本的预测值的确应该高于负样本。

1.6. TAR，FRR，FAR

这几个指标在人脸验证中被广泛使用，人脸验证即匹配两个人是否是同一个人，通常用特征向量的相似度进行描述，如果相似度概率大于阈值 T，则被认为是同一个人。

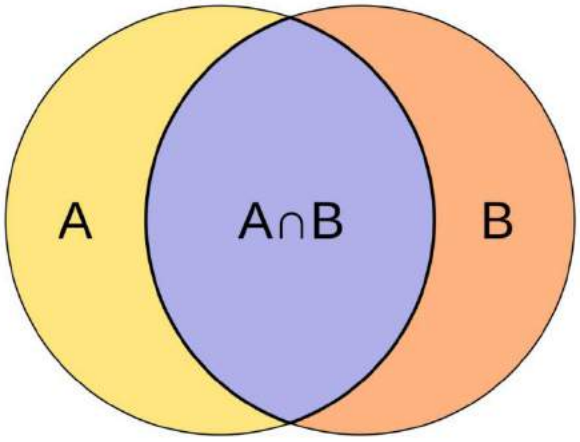
TAR（True Accept Rate）表示正确接受的比例，多次取同一个人的两张图像，统计该相似度值超过阈值 T 的比例。FRR（False Reject Rate）就是错误拒绝率，把相同的人的图像当做不同人的了，它等于 1-TAR。 与之类似，FAR（False Accept Rate）表示错误接受的比例，多次取不同人的两张图像，统计该相似度值超过 T 的比例。

增大相似度阈值 T，FAR 和 TAR 都减小，意味着正确接受和错误接受的比例都降低，错误拒绝率 FRR 会增加。减小相似度阈值 T，FAR 和 TAR 都增大，正确接受的比例和错误接受的比例都增加，错误拒绝率 FRR 降低。

2 检索与回归指标

2.1. IoU

IoU 全称 Intersection-over-Union， 即交并比，在目标检测领域中，定义为两个矩形框面积的交集和并集的比值，IoU=A∩B/A∪B。



如果完全重叠，则 IoU 等于 1，是最理想的情况。一般在检测任务中，IoU 大于等于 0.5 就认为召回，如果设置更高的 IoU 阈值，则召回率下降，同时定位框也越更加精确。

在图像分割中也会经常使用 IoU，此时就不必限定为两个矩形框的面积。比如对于二分类的前背景分割，那么 IoU=(真实前景像素面积∩预测前景像素面积)/(真实前景像

素面积∪预测前景像素面积)，这一个指标，通常比直接计算每一个像素的分类正确概率要低，也对错误分类更加敏感。

2.AP 和 mAP

Average Precision 简称 AP，这是一个在检索任务和回归任务中经常使用的指标，实际等于 Precision-Recall 曲线下的面积，这个曲线在上一小节已经说过，下面针对目标检测中举出一个例子进行计算，这一个例子在网上也是广泛流传。

假如一幅图像，有 10 个人脸，检索出来了 20 个目标框，每一个目标框的概率以及真实的标签如下，真实标签的计算就用检测框与真实标注框的 IoU 是否大于 0.5 来计算。

第一步，就是根据模型得到概率，计算 IoU 得到下面的表。

Id	Score	Label
1	0.23	0
2	0.76	1
3	0.01	0
4	0.91	1
5	0.13	0
6	0.45	0
7	0.12	1
8	0.03	0
9	0.38	1
10	0.11	0
11	0.03	0
12	0.09	0
13	0.65	0
14	0.07	0
15	0.12	0
16	0.24	1
17	0.1	0
18	0.23	0
19	0.46	0
20	0.08	1

第二步，将上面的表按照概率进行排序

Id	Score	label
4	0.91	1
2	0.76	1
13	0.65	0
19	0.46	0
6	0.45	0
9	0.38	1
16	0.24	1
1	0.23	0
18	0.23	0
5	0.13	0
7	0.12	1
15	0.12	0
10	0.11	0
17	0.1	0
12	0.09	0
20	0.08	1
14	0.07	0
8	0.03	0
11	0.03	0
3	0.01	0

Precision 的计算如下，以返回的 top-5 结果为例：

Id	Score	label
4	0.91	1
2	0.76	1
13	0.65	0
19	0.46	0
6	0.45	0
9	0.38	1
16	0.24	1
1	0.23	0
18	0.23	0
5	0.13	0
7	0.12	1
15	0.12	0
10	0.11	0
17	0.1	0
12	0.09	0
20	0.08	1
14	0.07	0
8	0.03	0
11	0.03	0
3	0.01	0

在这个例子中，true positives 就是真正的人脸，从 Label 一栏可以看出，指的是 id = 4，2，7，9，16，20 的样本。

前 5 个概率值最大的 id 中 13，19，6 是 false positives。所以此时的 Precision=2/5=40%，即选定了 5 个人脸，但是只有两个是对的。recall=2/6=33.3%，即总共有 6 个人脸，但是只召回了 2 个。

在一个实际的目标检测任务中，目标的数量不一定是 5 个，所以不能只通过 top-5 来来衡量一个模型的好坏，选定的 id 越多，recall 就越高，precision 整体上则会呈现出下降趋势，因为排在前面的概率高的，一般更有可能是真实的样本，而后面概率低的更有可能是负样本。

令 N 是所有 id，如果从 top-1 到 top-N 都统计一遍，得到了对应的 precision 和 recall，以 recall 为横坐标，precision 为纵坐标，则得到了检测中使用的 precision-recall 曲线，虽然整体趋势和意义与分类任务中的 precision-recall 曲线相同，计算方法却有很大差别。

在 PASCAL VOC 2010 年以前的比赛中，AP 的具体计算方法如下：

设置 11 个阈值[0, 0.1, 0.2, ... , 1]，计算 recall 大于等于每一个阈值时的最大 precision，AP 就是这 11 个值的平均值。根据上表中的计算方法，选择不同的 N 时会有不同的 precision 和 recall，所以也有可能有不同的 N 落在同样的 recall 区间，此时就需要选择其中最大的精度值，这时候曲线上的点就不一定对应同一个阈值时的 recall。

AP 就是这 11 个 precision 的平均值，将所有类别的 AP 再取平均，就得到了 mAP。

PASCAL VOC 2010 年提出了一个更好的指标，去除了 11 点的设定，对于样本不均衡的类的计算更加有效。

假设有 N 个 id，其中有 M 个 label，则取 M 个 recall 节点，从 0 到 1 按照 1/M 的等间距，对于每个 recall 值，计算出大于该 recall 值的最大 precision，然后对这 M 个 precision 值取平均得到最后的 AP 值，mAP 的计算方法不变。

AP 衡量的是学出来的模型在一个类别上的好坏，mAP 衡量的是学出的模型在所有类别上的好坏。

3 图像质量评价指标

图像在获取，压缩，存储，传输，解压缩，显示，甚至打印的过程中，都有可能受到环境的干扰造成质量下降。图像的质量，通常跟图像噪声，模糊，对比度，美学等有关系。

在图像质量评估的指标中，根据对原始无损图像的要求，通常分为三大类[1-2]，分别是**Full Reference Image Quality Assessment(FR-IQA)全参考图像评价，Reduced Reference Image Quality Assessment(FR-IQA)半参考图像评价，No Reference Image Quality Assessment(FR-IQA)无参考图像评价**。

Full Reference Image Quality Assessment(FR-IQA)全参考图像评价，需要原始的高质量的图像。常见的(FR-IQA)包括峰值信噪比 PSNR，结构一致性相似因子 structural similarity index measurement(SSIM)，视觉信息保真度 (Visual information fidelity, VIF)，视觉信噪比 (Visual signal-to-noise ratio, VSPR)，最显著失真 (Most apparent distortion, MAD)等。

Reduced Reference Image Quality Assessment(RR-IQA)半参考图像评价，不需要原始图像本身，但是需要一些特征，在卫星和遥感图像中被使用的较多。RR-IQA 类方法常常在不同的特征空间中使用，主要思路是对 FR-IQA 类评价指标进行近似。

No Reference Image Quality Assessment(NR-IQA)无参考图像评价，完全基于图像本身，不再需要原始图。不过由于没有原始的图像，需要对原始的图像进行统计建模，同时还要兼顾人眼的视觉特征，本来这就有一定的主观和不确定性。虽然研究人员提出了数十个 NR-IQA 指标，但是真正广泛使用的没有几个。另外无参考的美学质量评估也是当前比较开放的一个问题，它需要更多考虑摄影学等因素。

质量评价因子非常的多，本小节只介绍其中的 4 个，后续会专门撰写综述来讲解。

3.1.信噪比 SNR 与峰值信噪比 PSNR

信噪比，即 SNR(SIGNAL-NOISE RATIO)，是信号处理领域广泛使用的定量描述指标。它原是指一个电子设备或者电子系统中信号与噪声的比例，计量单位是 dB，其计算方法是 10*lg(Ps/Pn)，其中 Ps 和 Pn 分别代表信号和噪声的有效功率，也可以换算成电压幅值的比率关系：20*lg(Vs/Vn)，Vs 和 Vn 分别代表信号和噪声电压的“有效值”。

在图像处理领域，更多的是采用峰值信噪比 PSNR (Peak Signal to NoiseRatio)，它是原图像与处理图像之间均方误差(Mean Square Error)相对于(2^n-1)^2 的对数值，其中 n 是每个采样值的比特数，8 位图像即为 256。

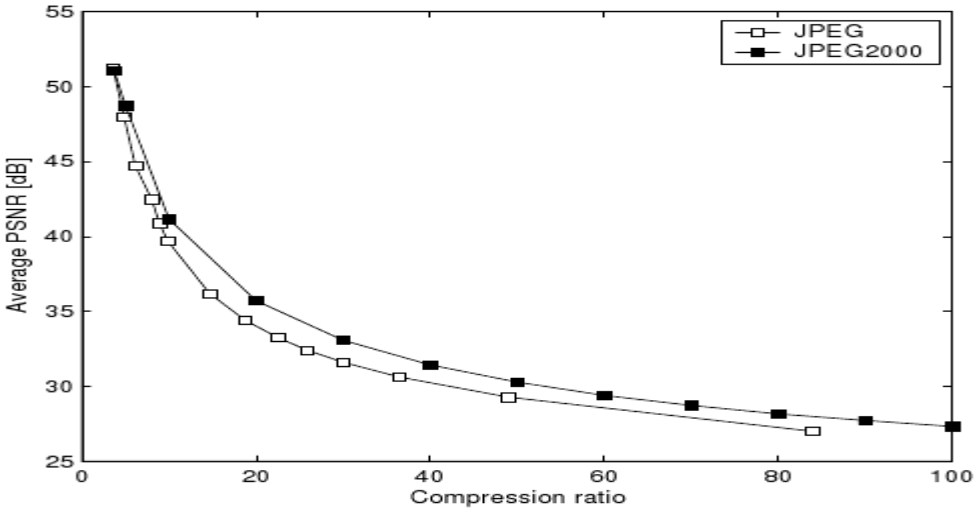
$$PSNR = 10 \times \log\left(\frac{255^2}{MSE}\right)$$

PSNR 越大表示失真越小，均方误差的计算如下：

$$MSE = \frac{\sum_{0\le i\le M}\sum_{0\le j\le N}(f_{i,j}-\mu_{i,j})^2}{M\times N}$$

其中 M，N 为图像的行与列数，μi,j 是像素灰度平均值，fi,j 即像素灰度值。

下面展示了 JPEG 和 JPEG2000 算法在不同压缩率下的 PSNR，通常 PSNR 大于 35 的图像质量会比较好。



3.2.结构一致性相似因子 SSIM

PSNR 从底层信噪的角度来评估图像的质量，但是人眼对质量的评价关注的层次其实更高。根据 Human visual system model，人眼观察图像有几个特点：

(1)低通过滤器特性，即人眼对于过高的频率难以分辨。(2)人眼对亮度的敏感大于对颜色的敏感。(3)对亮度的响应不是线性变换的，在平均亮度大的区域，人眼对灰度误差不敏感。(4)边缘和纹理敏感，有很强的局部观察能力。

structural similarity index measurement(SSIM)是一种建立在人眼的视觉特征基础上的用于衡量两幅图像相似度的指标，结果在 0～1 之间。

结构相似性理论认为自然图像信号是高度结构化的，空域像素间有很强的相关性并蕴含着物体结构的重要信息。它没有试图通过累加与心理物理学简单认知模式有关的误差来估计图像质量，而是直接估计两个复杂结构信号的结构改变，并将失真建模为亮度、对比度和结构三个不同因素的组合。用均值作为亮度的估计，标准差作为对比度的估计，协方差作为结构相似程度的度量。

PSNR 忽略了人眼对图像不同区域的敏感度差异，在不同程度上降低了图像质量评价结果的可靠性，而 SSIM 能突显轮廓和细节等特征信息。

SSIM 具体的计算如下：首先结构信息不应该受到照明的影响，因此在计算结构信息时需要去掉亮度信息，即需要减掉图像的均值；其次结构信息不应该受到图像对比度的影响，因此计算结构信息时需要归一化图像的方差。

通常使用的计算方法如下，其中 C1，C2，C3 用来增加计算结果的稳定性，光度 L，对比度 C，结构对比度 S 计算如下：

$$l(x,y)=\frac{2\mu_x\mu_y+C_1}{\mu_x^2+\mu_y^2+C_1}$$

ux，uy 为图像的均值

$$C(x,y)=\frac{2d_xd_y+C_2}{d_x^2+d_y^2+C_2}$$

dx，dy 为图像的方差

$$S(x,y)=\frac{d(x,y)+C_3}{d_xd_y+C_3}$$

d(x,y)为图像 x，y 的协方差。而图像质量 SSIM = L(x,y)^aC(x,y)^bS(x,y)^c，其中 a，b，c 分别用来控制三个要素的重要性，为了计算方便可以均选择为 1，C1，C2，C3 为比较小的数值，通常 C1=(K1×L)^2, C2=(K2×L)^2，C3 = C2/2， K1 << 1， K2 << 1，L 为像素的最大值(通常为 255)。当 a，b，c 都等于 1，C3=C2/2 时，SSIM 的定义式就为：

$$SSIM=\frac{(2\mu_x\mu_y+C_1)(2d_xd_y+C_1)}{(\mu_x^2+\mu_y^2+C_2)(d_x^2+d_y^2+C_2)}$$

SSIM 发展出了许多的改进版本，其中较好的包括 Fast SSIM，Multi-scale SSIM。

3.3.无参考锐化因子 CPBD

上面说的这两个，都是需要参考图像才能进行评价的，而 CPBD 即 cumulative probability of blur detection (CPBD)，是一种无参考的图像锐化定量指标评价因子。它是基于模糊检测的累积概率来进行定义，是基于分类的方法。

首先要说 Just-noticeable-distortion-model，简称 JND 模型，即恰可察觉失真模型，它建模人眼能够察觉的图像底层特征，只有超过一定的阈值才会被察觉为失真图像。如果在空间域计算，一般会综合考虑亮度，纹理等因素，比如用像素点处局部平均背景亮度值作为亮度对比度阈值，用各个方向的梯度作为纹理阈值。如果在变换域计算，则可以使用小波系数等，这里对其计算方法就不再详述。

在给定一个对比度高于 JND (Just Noticeable Difference)参考的情况下，定义 JNB (Just-noticeable-blue)指标为感知到的模糊像素的最小数目，边缘处像素的模糊概率定义如下：

$$P_{blur} = P(e_i) = 1 - \exp(-|\frac{\omega(e_i)}{\omega_{JNB}(e_i)}|^{\beta})$$

其中分子是基于局部对比度的 JNB 边缘宽度，而分母是计算出的边缘宽度。对于每一幅图像，取子块大小为 64×64，然后将其分为边缘块与非边缘块，非边缘块不做处理。对于每一个边缘块，计算出块内每个边缘像素的宽度。当 pblur<63%，该像素即作为有效的像素，用于计算 CPBD 指标，定义如下：

$$CPBD = P(P_{blur} \le P_{jnb}) = \sum_{P_{blur}=0}^{P_{blur}=P_{jnb}} P(P_{blur})$$

CPBD 的作者采用了高斯模糊的图像与 JPEG 压缩图像进行实验，表明 CPBD 是符合人类视觉特性的图像质量指标，值越大，所反映出的细节越清晰，模糊性越弱。因此，可以将此指标用于定量评判滤波后的图像的锐化质量。关于该指标和 PSNR 和 SSIM 的对比，大家可以去阅读笔者的硕士论文[3]。

图像质量评价这个领域的坑太大，水太多，如果只是感兴趣，就建议不要入了。

4 图像生成评价指标

当我们要评估一个生成模型的性能的时候，有 2 个最重要的衡量指标。

(1) 确定性：生成模型生成的样本一定属于特定的类别，也就是真实的图像，而且必须要是所训练的图片集，不能用人脸图像训练得到了手写数字。

(2) 多样性：样本应该各式各样，如果用 mnist 进行训练，在没有条件限制的条件下，应该生成 0，1，2，3…，而不是都是 0，生成的各个数字也应该具有不同的笔触，大小等。

除此之外，还会考虑分辨率等，因此评价生成模型也需要从这几个方向着手。

4.1.inception score

inception score 是最早的用于 GAN 生成的图像多样性评估的指标，它利用了 google 的 inception 模型来进行评估，背后的思想就完美满足上面的两个衡量指标。

Inception 图像分类模型预测结果是一个 softmax 后的向量，即概率分布 p(y|x)。一个好的分类模型，该向量分布的熵应该尽可能地小，也就是样本必须明确符合某一个类，其中的一个值很大，剩下的值很小。另外，如果把 softmax 后的向量组合并在一起

形成另一个概率分布 p(y)，为了满足多样性，这个分布的熵应该是越大越好，也就是各种类别的样本都有。

具体实现就是让 p(y|x)和 p(y)之间的 KL 散度越大越好，连续形式的表达如下。

$$\int_x p(y|x)[logp(y|x) - log \int_x p(y|x)dx]dx$$

实际的计算就是将积分换成求和：

$$\frac{1}{N}\sum_i^N p(y|x_i)\,[\log p((y|x_i)) - \log \sum_j^N p(y|x_j)]$$

Inception Score 是一个非常好的评价指标，它同时评估生成图像的质量和多样性，前段时间大火的 BigGAN，就是将 Inception Score 提升为原来最好模型的 3 倍不止。

不过 Inception Score 也有缺陷，因为它仅评估图像生成模型，没有评估生成的图像与原始训练图像之间的相似度，因此虽然鼓励模型学习了质量好，多样性好的图像，但是却不能保证是我们想要的图像。Mode 分数对其进行了改进，增加了 KL 散度来度量真实分布 P_r 与生成分布 P_g 之间的差异。

4.2.Kernel MMD

最大平均差异 maximum mean discrepancy Kernel 也是一个用于判断两个分布 p 和 q 是否相同的指标。它的基本假设就是如果两个样本分布相似，那么通过寻找在样本空间上的连续函数 f，求不同分布的样本 f 函数的均值，计算均值的差作为两个分布在 f 函数下的平均差异，选择其中最大值就是 MMD。

对于深度学习任务来说，可以选择各种预训练模型的特征空间，比如性能很好的 ResNet。

MMD 方法的样本复杂度和计算复杂度都比较低，不过是有偏的，关键就在于用于选择的函数空间是否足够丰富。

除了以上指标外，还有 Wasserstein 距离，Fréchet Inception 距离等，以后再开篇详述。

参考文献

[1] Yuan Y, Guo Q, Lu X, et al. Image quality assessment[J]. Neurocomputing, 2015: 227-241.

[2] Kamble V, Bhurchandi K M. No-reference image quality assessment algorithms: A survey[J]. Optik, 2015, 126(11): 1090-1097.
[3] 龙鹏. MRI 医学图像增强与分割新方法[D]. 中国科学院大学, 2015.
[4] Xu Q , Huang G , Yuan Y , et al. An empirical study on evaluation metrics of generative adversarial networks[J]. 2018.

5 总结

深度学习模型的评价方法实在是太多了，本文以计算机视觉领域为例，给大家介绍了最广泛使用的一部分指标。有些指标的设计是绝对没有争议的，有些指标的设计是带着主观性质的，不管怎么说，它们为大家的竞争和比较提供了指导。

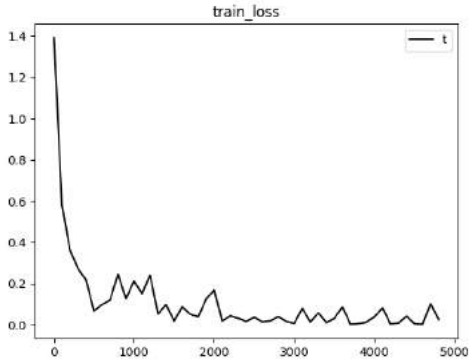
【AI 初识境】深度学习中常用的损失函数有哪些？

这是专栏《AI 初识境》的第 11 篇文章。所谓初识，就是对相关技术有基本了解，掌握了基本的使用方法。本文来说说深度学习中常见的损失函数(loss)，覆盖分类，回归任务以及生成对抗网络，有了目标才能去优化一个模型。

作者|言有三

1 什么是损失函数

在机器学习中，损失函数（loss function）是用来估量模型的预测值 f(x)与真实值 Y 的不一致程度，损失函数越小，一般就代表模型的鲁棒性越好，正是损失函数指导了模型的学习。



机器学习的任务本质上是两大类，分类问题与回归问题，再加上综合了判别模型和生成模型后在各类图像任务中大展拳脚的生成对抗网络，这一次我们就重点讲述这些内容。

2 分类任务损失

2.1、0-1 loss

0-1 loss 是最原始的 loss，它直接比较输出值与输入值是否相等，对于样本 i，它的 loss 等于：

$$L(y_i, f(x_i)) = \begin{cases} 0 \ if \ y_i = f(x_i) \\ 1 \ if \ y_i \neq f(x_i) \end{cases}$$

当标签与预测类别相等时，loss 为 0，否则为 1。可以看出，0-1 loss 无法对 x 进行求导，这在依赖于反向传播的深度学习任务中，无法被使用，0-1 loss 更多的是启发新的 loss 的产生。

2.2、熵与交叉熵 loss

在物理学有一个概念，就是熵，它表示一个热力学系统的无序程度。为了解决对信息的量化度量问题，香农在 1948 年提出了“信息熵”的概念，它使用对数函数表示对不确定性的测量。熵越高，表示能传输的信息越多，熵越少，表示传输的信息越少，我们**可以直接将熵理解为信息量**。

按照香农的理论，熵背后的原理是任何信息都存在冗余，并且冗余大小与信息中每个符号（数字、字母或单词）的出现概率或者说**不确定性**有关。概率大，出现机会多，则不确定性小，这个关系就用对数函数来表征。

为什么选择对数函数而不是其他函数呢？首先，不确定性必须是概率 P 的单调递降函数，假设一个系统中各个离散事件互不相关，要求其总的不确定性等于各自不确定性之和，对数函数是满足这个要求的。将不确定性 f 定义为 log(1/p)=-log(p)，其中 p 是概率。

对于单个的信息源，信源的平均不确定性就是单个符号不确定性-logpi 的统计平均值，信息熵的定义如下。

$$-\sum_{i=0}^{n} p_i \log\ p_i$$

假设有两个概率分布 p(x)和 q(x)，其中 p 是已知的分布，q 是未知的分布，则其交叉熵函数是两个分布的互信息，可以反应其相关程度。

从这里，就引出了分类任务中最常用的 loss，即 log loss，又名交叉熵 loss，后面我们统一称为交叉熵：

$$l(f,y) = -\sum_{i}^{n}\sum_{j}^{m} y_{ij} log f(x_{ij})$$

n 对应于样本数量，m 是类别数量，yij 表示第 i 个样本属于分类 j 的标签，它是 0 或者 1。对于单分类任务，只有一个分类的标签非零。f(xij) 表示的是样本 i 预测为 j 分类的概率。loss 的大小完全取决于分类为正确标签那一类的概率，当所有的样本都分类正确时，loss=0，否则大于 0。

2.3、softmax loss 及其变种

假如 log loss 中的 f(xij)的表现形式是 softmax 概率的形式，那么交叉熵 loss 就是我们熟知的 softmax with cross-entropy loss，简称 softmax loss，所以说 softmax loss 只是交叉熵的一个特例。

softmax loss 被广泛用于分类分割等任务，而且发展出了很多的变种，有针对不平衡样本问题的 weighted softmax loss， focal loss，针对蒸馏学习的 soft softmax loss，促进类内更加紧凑的 L-softmax Loss 等一系列改进，**公众号早在一年前就撰写过综述如下：**

【技术综述】一文道尽 softmax loss 及其变种

2.4、KL 散度

Kullback 和 Leibler 定义了 KL 散度用于估计两个分布的相似性，定义如下；

$$D_{kl}(p|q) = \sum_i p_i \log\left(\frac{p_i}{q_i}\right)$$

Dkl 是非负的，只有当 p 与 q 处处相等时，才会等于 0。上面的式子也等价于

$$D_{kl}(p|q) = \sum_i p_i \log p_i - p_i \log q_i = -l(p,p) + l(p,q)$$

其中 l(p,p)是分布 p 的熵，而 l(p,q)就是 p 和 q 的交叉熵。假如 p 是一个已知的分布，则熵是一个常数，此时 dkl(p|q)与 l(p,q)也就是交叉熵只有一个常数的差异，两者是等价的。同时值得注意的是，KL 散度并不是一个对称的 loss，即 dkl(p|q) != dkl(q|p)，KL 散度常被用于生成式模型。

2.5、Hinge loss

Hinge loss 主要用于支持向量机中，它的称呼来源于损失的形状，定义如下：

$$l(f(x), y) = \max\big(0, 1 - yf(x)\big)$$

如果分类正确，loss=0，如果错误则为 1-f(x)，所以它是一个分段不光滑的曲线。

Hinge loss 被用来解 SVM 问题中的间距最大化问题。

2.6、Exponential loss 与 Logistic loss

Exponential loss 是一个指数形式的 loss，它的特点就是梯度比较大，主要用于 Adaboost 集成学习算法中，定义如下：

$$l(f(x), y) = e^{-\beta y f(x)}$$

logistic loss 取了 Exponential loss 的对数形式，它的定义如下：

$$l(f(x),y)=\frac{1}{ln2}\ln\left(1+e^{-yf(x)}\right)$$

logistic loss 梯度相对变化更加平缓。

此外还有 sigmoid cross_entropy_loss，可以被用于多标签分类任务或者不需要创建类间竞争机制的分类任务，在 Mask RCNN 中就被用了。

$$l(x,y)=-\sum_i(y_i\log\left(f(x_i)\right)+(1-y_i)\log\left(1-f(x_i)\right))$$

以上就涵盖了大部分常用的分类任务损失，多半都是对数的形式，这是由信息熵的定义，参数似然估计的本质决定的。

3 回归任务损失

在回归任务中，回归的结果是一些整数或者实数，并没有先验的概率密度分布，常使用的 loss 是 L1 loss 和 L2 loss。

3.1、L1 loss

Mean absolute loss(MAE)也被称为 L1 Loss，是以绝对误差作为距离：

$$\mathrm{MAE}=\frac{1}{n}\sum_1^n|(y_i-\dot{y}_i)|$$

由于 L1 loss 具有稀疏性，为了惩罚较大的值，因此常常将其作为正则项添加到其他 loss 中作为约束。L1 loss 的最大问题是梯度在零点不平滑，导致会跳过极小值。

3.2、L2 loss

Mean Squared Loss/ Quadratic Loss(MSE loss)也被称为 L2 loss，或欧氏距离，它以误差的平方和作为距离：

$$\mathrm{MSE}=\frac{1}{n}\sum_1^n(y_i-\dot{y}_i)^2$$

L2 loss 也常常作为正则项。当预测值与目标值相差很大时，梯度容易爆炸，因为梯度里包含了 x?t。

3.3、L1 loss 与 L2 loss 的改进

原始的 L1 loss 和 L2 loss 都有缺陷，比如 L1 loss 的最大问题是梯度不平滑，而 L2 loss 的最大问题是容易梯度爆炸，所以研究者们对其提出了很多的改进。

在 faster rcnn 框架中，使用了 smooth L1 loss 来综合 L1 与 L2 loss 的优点，定义如下：

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 \ \ if\ |x| < 1 \\ |x| - 0.5 \ otherwise \end{cases}$$

在 x 比较小时，上式等价于 L2 loss，保持平滑。在 x 比较大时，上式等价于 L1 loss，可以限制数值的大小。

为了增强 L2 loss 对噪声(离群点)的鲁棒性，研究者提出了 Huber loss，定义如下：

$$L = \begin{cases} \frac{1}{2}\sum_1^n (y_i - \dot{y}_i)^2 \ \ for |y_i - \dot{y}_i| \le \delta \\ \delta \cdot (|y_i - \dot{y}_i| - \frac{1}{2}\delta) \end{cases}$$

Huber 对于离群点非常的有效，它同时结合了 L1 与 L2 的优点，不过多出来了一个 delta 参数需要进行训练。

除此之外还有 Log-Cosh Loss 等损失，大家可以自己了解，也欢迎补充。

从上面可以看出，L1/L2 各有优劣，设计一个通用的框架同时满足 L1/L2 损失的优点是研究重点，我见过的最夸张的是这样的。

$$\rho(x, \alpha, c) = \qquad (12)$$
$$\begin{cases} \frac{1}{2}(x/c)^2 & \text{if } \alpha = 2 \\ \log\left(\frac{1}{2}(x/c)^2 + 1\right) & \text{if } \alpha = 0 \\ 1 - \exp\left(-\frac{1}{2}(x/c)^2\right) & \text{if } \alpha = -\infty \\ \frac{|2-\alpha|}{\alpha}\left(\left(\frac{(x/c)^2}{|2-\alpha|} + 1\right)^{(\alpha/2)} - 1\right) & \text{otherwise} \end{cases}$$

3.4、perceptual loss

对于图像风格化，图像超分辨率重建等任务来说，早期都使用了图像像素空间的 L2 loss，但是 L2 loss 与人眼感知的图像质量并不匹配，恢复出来的图像往往细节表现不好。

现在的研究中，**L2 loss 逐步被人眼感知 loss 所取代。人眼感知 loss 也被称为 perceptual loss（感知损失）**，它与 MSE 采用图像像素进行求差的不同之处在于所计算的空间不再是图像空间。

研究者们常使用 VGG 等网络的特征，令 ϕ 来表示损失网络，Cj 表示网络的第 j 层，CjHjWj 表示第 j 层的特征图的大小，感知损失的定义如下：

$$\text{loss} = \frac{1}{C_j H_j W_j} ||\varphi_j(y) - \varphi_j(\dot{y})||_2^2$$

可以看出，它有与 L2 loss 同样的形式，只是计算的空间被转换到了特征空间。

4 生成对抗网络损失

生成对抗网络即 Generative Adversarial Networks，简称 GAN，它是 2014 年以后兴起的无监督学习网络，现在有非常多的解读了，我们一年前也解读过，欢迎移步，适合初学者。

【技术综述】有三说 GANs（上）

原始的用于生成图片的 GAN 的损失函数包括了生成式模型和判别式模型两部分，如今 GAN 被用于各类任务，其他的各种损失也加入了进来，不过我们这里还是专门针对 GAN 的基本损失进行讲述。

4.1、GAN 的基本损失

GAN 是在生成模型和判别模型的相互博弈中进行迭代优化，它的优化目标如下：

$$\min \max V(D,G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{x \sim p_z(z)}[1 - \log D(G(z))]$$

从中可以看出，包括两个部分，Ex ～ pdata(x)[logD(x)] 和 Ez ～ pz(z)[log(1?D(G(z)))]要求最大化判别模型对真实样本的概率估计，最小化判别模型对生成的样本的概率估计，生成器则要求最大化 D(G(z))，即最大化判别模型对生成样本的误判，这个 loss 是对数 log 的形式。

原始的 GAN 的损失使用了 JS 散度，两个分布之间越接近，它们的 JS 散度越小，但实际上这并不适合衡量生成数据分布和真实数据分布的距离，相关的分析已经非常的多了，本文如果展开就太长了，因此直接给解决方案。

4.2、-log D trick

Ian Goodfellow 提出了-log D trick，即把生成器 loss 改成如下，使得生成器的损失不依赖于生成器 G

$$E_{x \sim p_g}[-\log D(x)]$$

这个等价最小化目标存在两个严重的问题。第一是它同时要最小化生成分布与真实分布的 KL 散度，却又要最大化两者的 JS 散度，这是矛盾的会导致梯度不稳定。第二，

因为 KL 散度不是对称的，导致此时 loss 不对称，对于正确样本误分和错误样本误分的惩罚是不一样的。第一种错误对应的是“生成器没能生成真实的样本”，即多样性差，惩罚微小；第二种错误对应的是“生成器生成了不真实的样本”，即准确性低，惩罚巨大。这样造成生成器生成多样性很差的样本，出现了常说的模式崩塌(collapse mode)问题。

4.3、Wasserstein GAN(简称 wgan)等改进方案

wgan 采用了 Earth-Mover 距离(EM 距离)作为 loss，它是在最优路径规划下的最小消耗，计算的是在联合分布 γ 下，样本对距离的期望值：

$$\mathbb{E}(x,y)\sim\gamma[||x-y||]$$

与原始的 GAN 的 loss 形式相比，其实 wgan 就是生成器和判别器的 loss 不取 log。wessertein 距离相比 KL 散度和 JS 散度的优势在于，即使两个分布的支撑集没有重叠或者重叠非常少，仍然能反映两个分布的远近。而 JS 散度在此情况下是常量，KL 散度可能无意义。

wgan 有一些问题，wgan-gp 改进了 wgan 连续性限制的条件，后面还有一些研究，大家可以自行跟进，我们后面也会讲述。

4.4、LS-GAN

LS-GAN 即 Least Squares Generative Adversarial Networks。它的原理部分可以一句话概括，即使用了最小二乘损失函数代替了 GAN 的损失函数，相当于最小化 P 和 Q 之间的 Pearson 卡方散度（divergence），这属于 f-divergence 的一种，有效地缓解了 GAN 训练不稳定和生成图像质量差多样性不足的问题。作者认为使用 JS 散度并不能拉近真实分布和生成分布之间的距离，使用最小二乘可以将图像的分布尽可能的接近决策边界，其损失函数定义如下：

$$\min\max V(D,G)=\frac{1}{2}E_{x\sim p_{data}(x)}[(\mathrm{D}(\mathrm{x})-b)^2]+\frac{1}{2}E_{x\sim p_z(z)}[\mathrm{D}(\mathrm{G}(\mathrm{z})-\mathrm{a})^2)]$$

以交叉熵作为损失，它的特点是会使得生成器不会再优化那些被判别器识别为真实图片的生成图片，即使这些生成图片距离判别器的决策边界仍然很远，也就是距真实数据比较远，这意味着生成器的生成图片质量并不高。而要想最小二乘损失比较小，则在混淆判别器的前提下还得让生成器把距离决策边界比较远的生成图片拉向决策边界，这就是 LS-GAN 的优势。

4.5、Loss-sensitive-GAN

在原始的 GAN 的损失函数后添加了一个约束项来直接限定 GAN 的建模能力，它的损失函数如下：

$$s(\theta,\emptyset^*) = E_{x\sim p_{data}} L_\theta(x) + \lambda E_{x\sim p_{data}, Z_g\sim P_g}(\Delta(x, Z_g) + L_\theta(x) - L_\theta(Z_g))_+$$

优化将通过最小化这个目标来得到一个“损失函数”（下文称之为 L 函数）。L 函数在真实样本上越小越好，在生成的样本上越大越好。它是以真实样本 x 和生成样本的一个度量为各自 L 函数的目标间隔，把 x 和生成样本分开。好处是如果生成的样本和真实样本已经很接近，就不必要求他们的 L 函数有个固定间隔，因为生成的样本已经很好。这样就可以集中力量提高那些距离真实样本还很远，真实度不那么高的样本，能更合理地使用 LS-GAN 的建模能力，被称为“按需分配”。

关于 GAN 的损失优化，这是一个不小的研究领域，下面是一个简单的汇总。

Name	Paper Link	Value Function
GAN	Arxiv	$L_D^{GAN} = E[\log(D(x))] + E[\log(1 - D(G(z)))]$ $L_G^{GAN} = E[\log(D(G(z)))]$
LSGAN	Arxiv	$L_D^{LSGAN} = E[(D(x) - 1)^2] + E[D(G(z))^2]$ $L_G^{LSGAN} = E[(D(G(z)) - 1)^2]$
WGAN	Arxiv	$L_D^{WGAN} = E[D(x)] - E[D(G(z))]$ $L_G^{WGAN} = E[D(G(z))]$ $W_D \leftarrow clip\_by\_value(W_D, -0.01, 0.01)$
WGAN_GP	Arxiv	$L_D^{WGAN\_GP} = L_D^{WGAN} + \lambda E[(\lvert\nabla D(\alpha x - (1-\alpha G(z)))\rvert - 1)^2]$ $L_G^{WGAN\_GP} = L_G^{WGAN}$
DRAGAN	Arxiv	$L_D^{DRAGAN} = L_D^{GAN} + \lambda E[(\lvert\nabla D(\alpha x - (1-\alpha x_p))\rvert - 1)^2]$ $L_G^{DRAGAN} = L_G^{GAN}$
CGAN	Arxiv	$L_D^{CGAN} = E[\log(D(x,c))] + E[\log(1 - D(G(z),c))]$ $L_G^{CGAN} = E[\log(D(G(z),c))]$
infoGAN	Arxiv	$L_{D,Q}^{infoGAN} = L_D^{GAN} - \lambda L_I(c,c')$ $L_G^{infoGAN} = L_G^{GAN} - \lambda L_I(c,c')$
ACGAN	Arxiv	$L_{D,Q}^{ACGAN} = L_D^{GAN} + E[P(class = c\lvert x)] + E[P(class = c\lvert G(z))]$ $L_G^{ACGAN} = L_G^{GAN} + E[P(class = c\lvert G(z))]$
EBGAN	Arxiv	$L_D^{EBGAN} = D_{AE}(x) + \max(0, m - D_{AE}(G(z)))$ $L_G^{EBGAN} = D_{AE}(G(z)) + \lambda \cdot PT$
BEGAN	Arxiv	$L_D^{BEGAN} = D_{AE}(x) - k_t D_{AE}(G(z))$ $L_G^{BEGAN} = D_{AE}(G(z))$ $k_{t+1} = k_t + \lambda(\gamma D_{AE}(x) - D_{AE}(G(z)))$

如果你对 GAN 还有更多兴趣，那就看这个参考网址吧，**https://hollobit.github.io/All-About-the-GAN/，不多不多，也就几千篇文章，**我大概看了 1000 篇的摘要，等闲下来再跟大家搞 GAN，是 Generative Adversarial Networks 噢。

5 总结

本文讲述了深度学习领域中常见的损失，学习灵活运用和设计损失本来不是初识境界的要求，不过还是让大家先有个基本感知吧。

【AI 初识境】给深度学习新手做项目的 10 个建议

这是专栏《AI 初识境》的第 12 篇文章。所谓初识，就是对相关技术有基本了解，掌握了基本的使用方法。在成为合格的深度学习算法工程师，尤其是工业界能够实战的调参选手之前，总会踏足很多的坑。本文就来说说那些需要掌握的基本技巧，如何避开那些新手常见的坑，以计算机视觉中的图像分类任务为例。

请注意，我这篇文章不是教你如何调参，而是教你不要在调参之前胡搞。

作者|言有三

1 项目开发之前应该做什么

在你真正开始撸代码之前，送大家一句话，“磨刀不误砍柴工”。拿到一个任务的时候，先不要上来就开始训模型，而是做好三件事。

1.1、知道你要做的任务是一个什么任务。

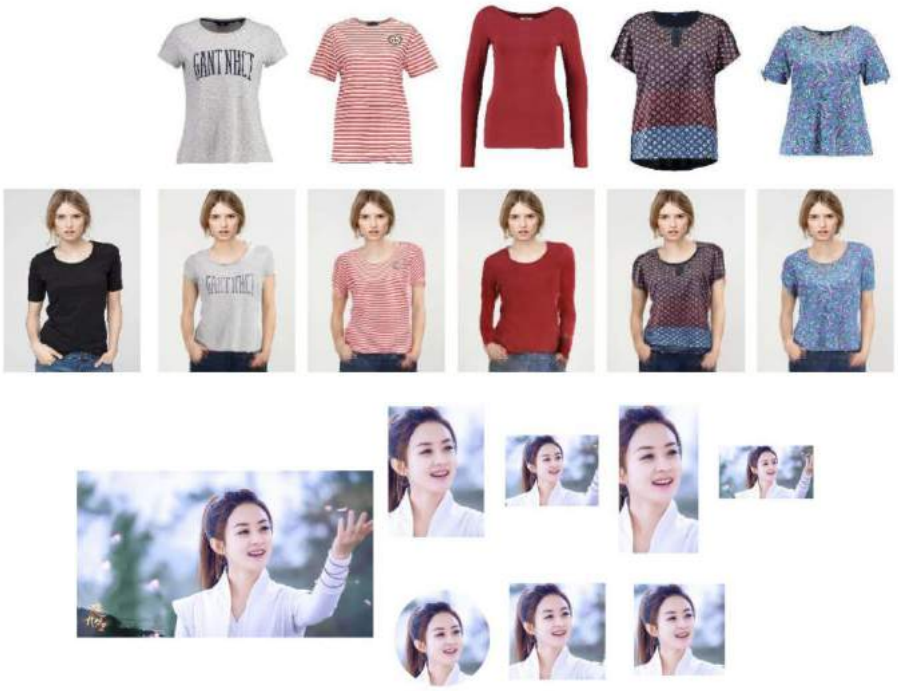
以图像分类为例，猫狗分类是一个分类任务，随便找个什么模型都能完成。

鸟类分类也是一个分类任务，但是不简单，不是随随便便拿个模型就能搞定。

表情识别最终也是一个分类，但是这不仅仅是一个分类问题，而是检测+分类问题。

产品经理们只会告诉你要实现什么功能，而不会告诉你用什么方案，对于简单任务来说这可能不需要思考，但是复杂任务一定要先充分调研认识。

你说上面的这个例子很简单，一眼就明白，那么我再举出几个例子，你是否一下子就能明白背后的核心技术？



1.2、找到竞争对手，做好预期。

比如你要做一个表情识别 API，要做一个美颜算法，一定要先看看你的竞争对手做的怎么样了，就算最后你做出来跟别人还差十万八千里，也不至于到最后一刻才发觉。**这叫知人之明和自知之明，一定要先有。**

在这个过程中，基本上就能确定要走的路，**第一条路是追随模仿别人，第二条路是超越别人**，这两者是不一样的。

前者，只要把已有成熟的资料收集到位，经验足够丰富，必定能成功，否则就是个人能力和资源问题。这样的一条路，相信老大们会给你布置一个明确的时间节点。

比如表情识别，很成熟，那你做出来也不能比别人差太多。

如果是第二条路，那么就意味着没有参考者，或者参考者做的也不行。那么最难的是什么，就是预期能做到什么水平。

可能技术已经成熟了，没人做，恭喜你，赶紧搞。

可能技术比较前沿，能做技术储备，但是还无法落地。

可能根本就还做不了。

1.3、想好你需要什么样的数据，从源头上降低任务的难度。

在公司干了四年活，大部分项目都需要自己准备数据，不会有现成的数据可以使用，而如何准备数据，这是需要经验的。

举个最简单的例子，假如我们需要开发一个分类算法来分析一张图片中的人是不是在笑，图像可能是这样。

你会爬取或者从数据集中拿到很多不同表情的数据然后就直接开干吗？显然那是错误的路子。

开始一个任务后我们首先就应该想怎么降低任务的难度，对于此任务，起作用的只有嘴唇这块区域，那么我们完全可以基于嘴唇区域来训练一个分类模型。

高精度成熟的人脸检测和关键点检测算法是很多的，所以你可能需要准备的训练数据是这样的。当然，如果你直接基于关键点的结果来做也是可以的，这就回到了第一个问题了。

这样至少有两个好处，(1) 明显这个分类更加简单了。(2) 可以使用更小的输入图完成任务，计算代价也更低。

2 训练模型从哪里开始

很少有一个任务可以拿现有的模型使用，你通常是需要训练自己的模型的，那么在训练一个模型的时候，应该怎么样开始。

这里牵涉到 3 个问题，框架，基准模型和数据，其中任何一环都有可能有问题。

2.1、首先确定框架

没有一个深度学习任务不需要一个框架来训练，很多的时候你不得不在不同的框架之间进行切换。比如做分类分割或者检测，caffe 都很好用。做风格化搞 GAN，就得上 tensorflow 或者 pytorch 了，你一定要先选择一个工具，不然很可能会陷入找到了很多中 github 方案，这个试了遇到困难换下一个，下一个又遇到困难。

有自己最拿手的一个框架，选定项目就坚定地干，遇到了问题就去解决。

关于框架，大家可以从我们的系列文章中开始快速上手。

- 【Keras 速成】Keras 图像分类从模型自定义到测试
- 【paddlepaddle 速成】paddlepaddle 图像分类从模型自定义到测试
- 【pytorch 速成】Pytorch 图像分类从模型自定义到测试
- 【tensorflow 速成】Tensorflow 图像分类从模型自定义到测试
- 【caffe 速成】caffe 图像分类从模型自定义到测试
- 【mxnet 速成】mxnet 图像分类从模型自定义到测试
- 【cntk 速成】cntk 图像分类从模型自定义到测试
- 【DL4J 速成】Deeplearning4j 图像分类从模型自定义到测试
- 【chainer 速成】chainer 图像分类从模型自定义到测试
- 【MatConvnet 速成】MatConvnet 图像分类从模型自定义到测试

2.2、然后确定基准模型

最终工业级部署的时候，你往往需要一个效率更高的模型。但是除非你是老司机，否则不应该在还没有确定方案是否可行的时候就想自己的模型，而是应该**从一个绝对可靠的模型开始**，比如 resnet18，比如 mobilenet，正确地使用好它们，得到还不错的结果，将它的结果作为你自己算法要 PK 的对象。

关于基准模型，我们已经把最主要的模型全部解读了一遍，可以从中选择。

- 【完结】总结 12 大 CNN 主流模型架构设计思想
- 【模型解读】从 LeNet 到 VGG，看卷积+池化串联的网络结构
- 【模型解读】network in network 中的 1*1 卷积，你懂了吗
- 【模型解读】GoogLeNet 中的 inception 结构，你看懂了吗
- 【模型解读】说说移动端基准模型 MobileNets
- 【模型解读】pooling 去哪儿了？
- 【模型解读】resnet 中的残差连接，你确定真的看懂了？
- 【模型解读】“不正经”的卷积神经网络
- 【模型解读】从“局部连接”回到“全连接”的神经网络
- 【模型解读】深度学习网络只能有一个输入吗
- 【模型解读】“全连接”的卷积网络，有什么好？
- 【模型解读】从 2D 卷积到 3D 卷积，都有什么不一样
- 【模型解读】浅析 RNN 到 LSTM
- 【模型解读】历数 GAN 的 5 大基本结构

2.3、最后准备好数据

在前面你想好自己需要的数据了，接下来就是去采集到这些数据。完成一个项目，就是不断迭代模型和数据的过程。

刚开始的时候，你不需要数据都到位，但是要考虑好以下因素。这里我们不管数据是自己采集的还是从公开数据集中获取的。

(1) 准备大小合适的数据量。以二分类任务为例，你不能拿 500 张图就开始干，没任何意义。你不能苛求一开始就有 50000 万数据，也不合理，万一搞失败了还浪费资源。笔者

先后做过 10 余个分类任务，完成任务上线从 3000 到 10 万都用过，我的建议是，尽量先准备个 3000 数据再开干，不然就太没诚意了。

(2) 从简单数据开始。以人脸识别为例，各种公开数据集不要混着用，分布不一致难度也不相同，你应该先专注简单的属于同一个分布的，比如找 10000 个正脸或者姿态小的数据，方案验证通过之后再增加难度。

很多的时候，你还要考虑覆盖各种场景(光照，背景)等。以上的这些东西，书不会告诉你，培训也没法教你，需要的是自己的积累，厚积薄发。

培养对数据的敏感性，可以从咱们的数据相关的文章开始看。

- 【数据】深度学习从“数据集”开始
- 【技术综述】深度学习中的数据增强（下）
- [综述类] 一文道尽深度学习中的数据增强方法（上）

3 正确训练模型的基本常识

终于讲到训练模型了，程序员干活从来不会一帆风顺，你的小模型不会这么听话，以下是一些必须要注意的事项，不管用什么框架都适用。

3.1 注意网络的输入大小

你极有可能是从 finetune 其他的模型开始，以图像分类为例，公开的模型大多是以 224*224 为尺度，而你的任务未必也需要这样。

如果你想要区分不同种类的鸟，那么因为细节在鸟的局部身体部位，你的输入恐怕是要更大一些才能提取到好的特征，比如放大一倍，用 448*448，这也是论文中常用的。

如果你只是要区分不同表情，用上了上面的嘴唇数据，那 224*224 纯属浪费，你可能只需要一个 48*48 的输入就足以很高准确率完成任务。

对于图像分割和目标检测，标准又不太一样。究竟使用多大的输入，这需要你依靠经验来确定，而且还和能给你多少资源，以及自己优化模型的能力相关。

3.2 注意特征输出大小

前面说了输入，这里再说输出，包括最后一层卷积的大小和通道数等。

首先看大小，对于一个分类任务来说，最后卷积层抽象为一个 k*k 的特征图，然后进行池化，全连接。如果这个特征图太大，知识根本就没有抽象出来，如果太小，表征能力又可能不够。

根据不同的难度，3*3，5*5，7*7 我都用过，但是没有用过 9*9 以上的，试过分类性能会下降，计算量还增加。imagenet 竞赛的那些网络基本上都是 7*7，兼顾了性能。

这个输出大小，就由**输入大小和网络的全局 stride** 来决定，不断遇到很多同学没有注意这个问题，结果模型性能很差的，这是基本素质。

3.3 正确地使用数据

前面已经准备了一些好的数据，别再用的时候却搞错了，对于图像分类来说有以下几个准则。

(1) 随机打乱你的数据。

否则每个 batch 给的是同样类别的数据，不一定能保证模型学的正常。

(2) 在线做一些基本的数据预处理和增强。

图像缩放操作，你要想好是用有变形的缩放，即统一缩放到固定大小。还是等比例缩放，即把短边缩放到一定尺度。

裁剪操作，随机裁剪是最简单有效的数据增强方案，一开始就可以做起来，比如 256*256 随机裁剪 224*224，不必要过于复杂。

镜像操作，也就是 mirror 参数，不是所有的任务都可以翻转的，根据自己任务使用。

减均值和归一化操作，其实这一步倒并非必要，因为网络自然可以学习到这一点。不过你做一下，通常不会有副作用。

更多的数据增强先不要急着做，因为那会增加网络优化的难度和时间。

这些操作都要在线做而不是离线准备好存入本地文件，这是很低效的。

3.4 正确地进行训练

接下来就开始训练，你可能会遇到各种与自己期望不相符的结果，其中一些很可能是你自己的错误造成的，因此有一些基本的训练参数需要注意。

(1) 用好学习率策略。

如果你没有经验，就不要一开始就使用 SGD，虽然它可能取得更好的结果。直接用 Adam，并且使用它的默认参数 m1=0.9，m2=0.999，lr=0.001，学习率可以调整，其他两个参数基本不需要动。更多比较可以参考我们之前的文章。如果你的学习率搞的不好，很可能出现梯度爆炸或者不收敛。

- 【模型训练】如何选择最适合你的学习率变更策略
- 【模型训练】SGD 的那些变种，真的比 SGD 强吗

(2) 正确使用正则项。

weight decay 是一个非常敏感的参数，如果你不是很有经验，从一个很小或者为 0 的值开始。

训练的时候可以用 dropout，测试的时候是不需要用的。

(3) 正确使用 BN 层。

Batch Normalization 是一个好东西，加快训练速度降低过拟合，但是你要注意它在训练的时候和测试的时候是不一样的。

use_global_stats 这个参数在训练时是 false，测试时是 true，如果你没用对，那么可能训练无法进行，或者测试结果不对。

在训练过程中，你可能会遇到各种各样奇葩的问题。

比如网络 loss 不正常，怎么调都不管用。

比如训练好好的，测试就是结果不对。

bug 天天有，深度学习算法工程师遇到的特别多，如果你想交流更多，就来有三 AI 知识星球实时提问交流吧，大咖众多，总有能解决你问题的。

5 总结

初识境界到此基本就结束了，这一系列是为大家奠定扎实的深度学习基础，希望学习完后大家能有收获。

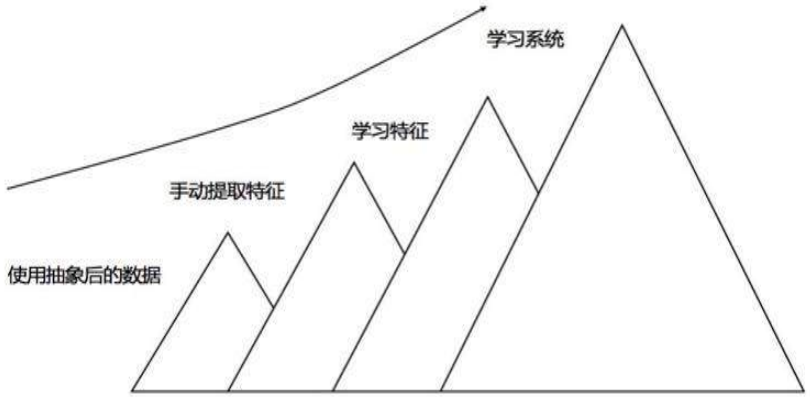
【AI 不惑境】数据压榨有多狠，人工智能就有多成功

从本文开始就进入了专栏《AI 不惑境》的更新了，这是第一篇文章，讲述数据如何驱动深度学习。进入到不惑境界，就是向高手迈进的开始了，在这个境界需要自己独立思考。如果说学习是一个从模仿，到追随，到创造的过程，那么到这个阶段，应该跃过了模仿和追随的阶段，进入了创造的阶段。从这个境界开始，讲述的问题可能不再有答案，更多的是激发大家一起来思考。

作者|言有三

深度学习成功源于三驾马车，**模型，数据和硬件**，这背后最核心的还是数据，深度学习正是因为学会了从数据中抽象知识，才能够完成各种各样的任务。

人工智能的发展，伴随着对数据的使用方法的进化，今天就来聊聊。



1 数据与学习

我一直对学生说，如果你不能认识到数据对一个任务的重要性，不知道什么样的数据能够完成手中的任务，就不算真正的入门深度学习。

在此之前，你可以去沉迷于各种框架，技巧，项目。

我们回想一下，大部分人的成长是什么样的过程。

(1) 一个刚刚出生的小孩，对这个世界的一切都表现出了无差别的兴趣，接受各种信息就是成长。

(2) 在青少年时期，我们在家长和老师的带领下，从背课文写作业开始学习，大部分错误的行为会得到纠错，正确的行为会得到奖励。

(3) 随着成长，有的人在自己工作的领域里熟练玩转数据和模型，充分发掘和运用已有的知识，另一部分人所做的事情不再有答案，需要自己去探索新的规律，比如成立自己的公司，创作新的知识。

这几个阶段，背后的核心都是数据。

（1）没有知识的时候，**所有已有的数据**都是知识。

（2）学习知识的时候，需要针对自己要学习的领域进行已有数据库的选择，想学语言就要背单词库，学数学就要做题库，学音乐就要练乐谱，这时候**用已有的数据进行学习**。

（3）使用知识的时候，就要调整自己学习到的知识**用于新输入的数据**，在这个过程中，知识也随之更新。

（4）创造知识的时候，就要观察社会和科学规律，从中进行总结，面对的就是**没有人整理过的数据**。

可以毫不夸张的说，人一生大部分时间都用着统计学获取，整理和分析数据，知识从数据中来，就像老子说的“道法自然”。

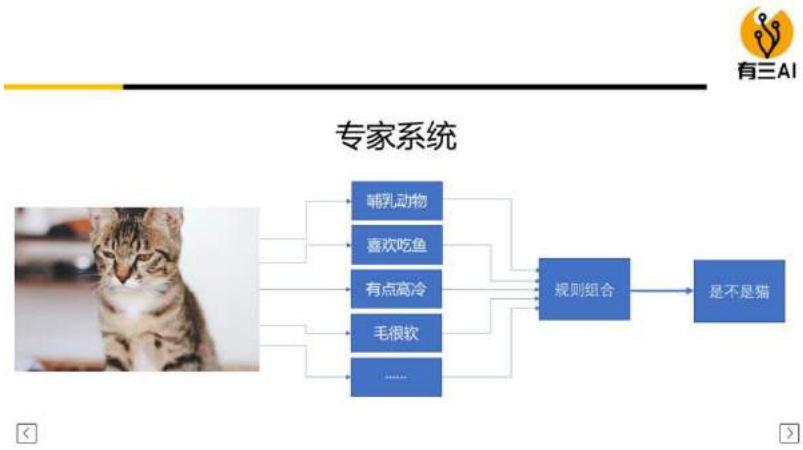
2 有监督特征工程到无监督特征学习

说起无监督和有监督方法，仍然先举一个依法治国和无为而治的对比。

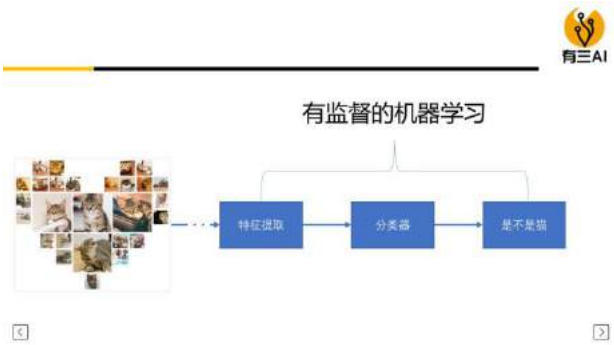
依法治国核心就在于设定了各种各样的法令让大家遵循，而无为而治的核心就是不干预，让国家在自然规律下运转。很明显后者是更高级的存在，也更难实现，不确定性大。

这个例子说的正是有监督和无监督方法在社会学的代表，从有监督到无监督是进步的，然后我们再看看智能系统的成长。

（1） 最初级的智能系统，其实就是**用机器来使用专家的知识**，依靠的是专家在某一个领域的大量的经验积累。从 20 世纪 60 年代开始到 80 年代第二次人工智能浪潮，专家系统的研究是非常流行的，大家感兴趣可以去了解。

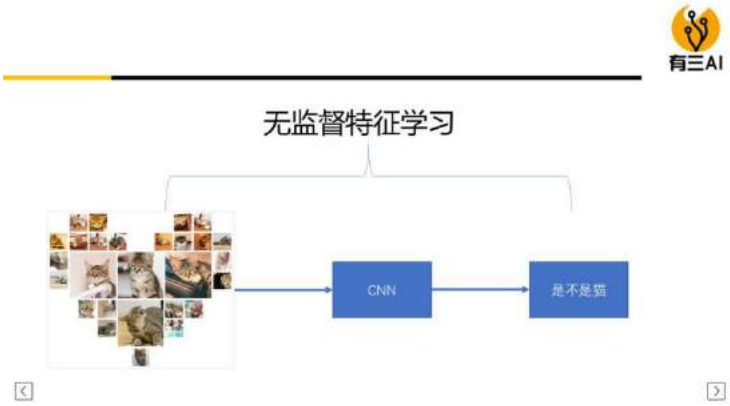


（2） 随着技术的发展，研究者发现专家系统实在是过于简单和脆弱，于是研究出了一系列的模型，包括人工神经网络/SVM 等等。**通过专家的经验对数据进行预处理，完成知识的初步抽象(提取特征)，之后丢给模型进行进一步的学习**。与专家系统相比模型的复杂度大大提升，因此也可以开始解决更加复杂的问题，比如人脸的检测，语音的识别。在 20 世纪末和 21 世纪初，有监督的机器学习方法得到了非常广泛的应用和研究。

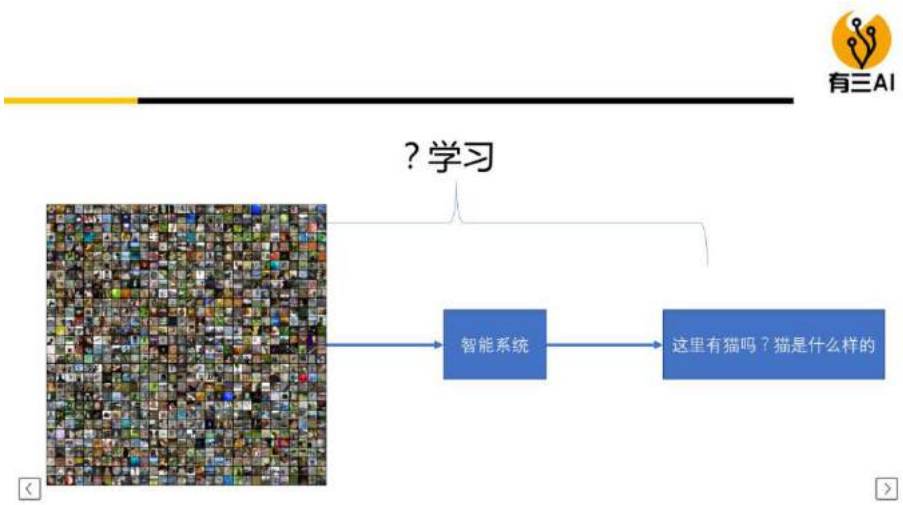


(3) 随着大数据的爆发以及科学家的不断探索，研究人员开始认识到通过专家的经验对数据进行预处理是不合适的，数据的维度太高，专家不可能知道每一个任务到底需要怎样的预处理，所以无监督特征学习方法诞生。对于一个无监督的特征学习系统，它的输入应该尽可能是原始的数据，最大程度上保证信息的完整。至于学习的规则，仍然由专家来制定。

于是专家设计出各种各样的模型架构和优化目标来指导系统从数据中进行学习，与有监督的特征工程的最大区别在于使用数据的方式，这一类方法也被称为特征学习，于是我们有了传统的机器学习算法和深度学习算法之分。



(4) 再往后发展，就需要机器自己创造模型，人类专家在其中所起的作用很小，甚至没有，这也是人工智能的未来，或许社会发展到一定的阶段，真的会有创造生命的那一天吧。



3 深度学习第一阶段-学习特征

在深度学习发展的第一阶段中，重点就是专家设计模型和优化策略，从数据中学习特征表达。

深度学习的成功很大程度上归功于卷积神经网络 CNN 模型架构，在图像，语音等领域都取得了大大突破。CNN 是一种无监督的特征学习模型，输入原始数据，然后完成学习。关于 CNN 的基础，大家可以去阅读公众号的相关文章。

在这个过程中，模型的架构固然会影响最终的结果，但是更重要的却是数据集，没有一个好的数据集，怎么都不可能训练出好的模型。关于数据集的重要性，可以阅读往期文章(点击图片)。

4 深度学习第二阶段-学习模型

在深度学习发展的第二阶段中，重点就是学习网络模型本身和各种相关的策略。

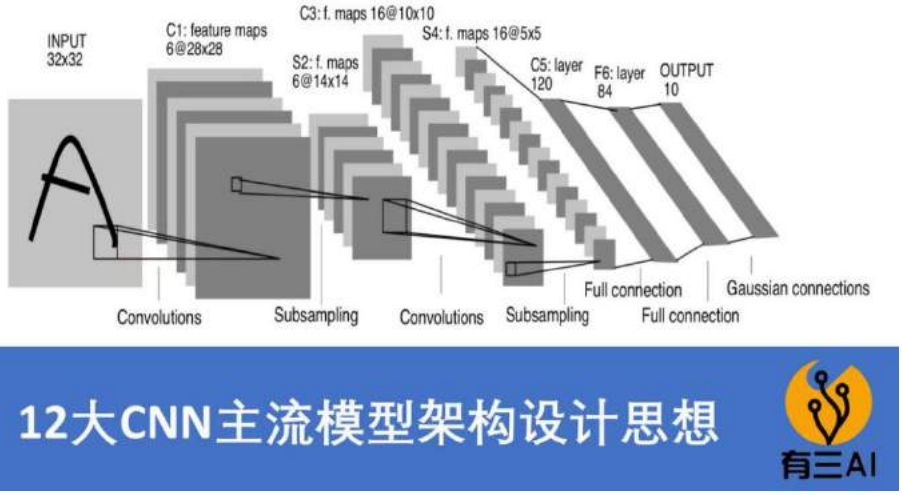
在第一阶段，典型的工作流程是准备数据，选择模型框架，定义各类优化参数，然后开始训练。

模型的架构需要研究人员手动设计，模型的各类训练参数包括归一化方法，初始化方法，激活函数等等也需要研究人员根据经验进行调试。数据的使用，包括预处理，增强策略也需要研究人员进行尝试。

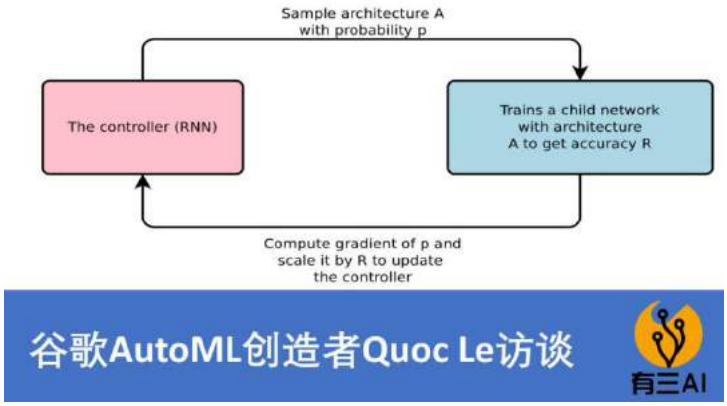
但是技术发展到今天，研究人员开始从数据中学习模型本身。

4.1、AutoML 自动模型结构设计技术

在深度学习发展的这些年里，研究人员用尽了各种手段去探索和设计各种各样的网络，研究网络的深度，宽度，卷积的方式，浅层深层的信息流动和融合等，可以参见往期文章(点击图片)。



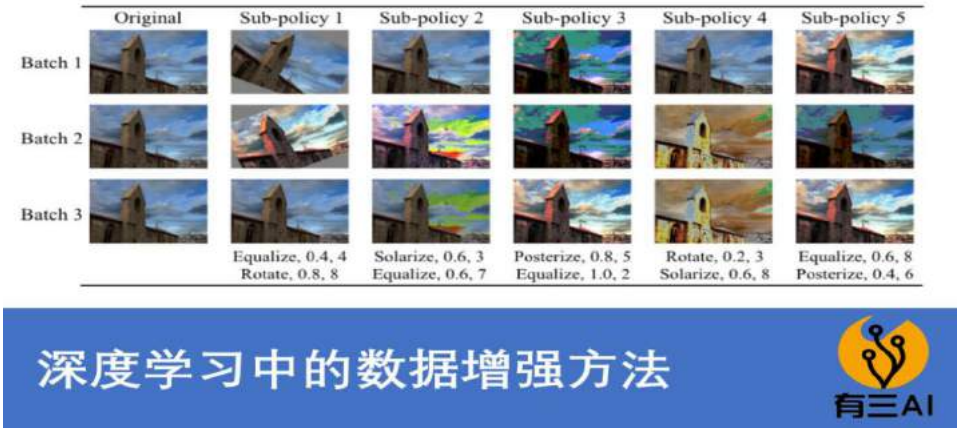
然而到了今天，新的网络设计方法开始流行，以 Google Brain 提出的 AutoML 为代表的技术，让**机器根据不同的任务(数据)，自动搜索最佳的模型架构，数据驱动了模型的学习。**



4.2、AutoAugment 自动数据增强策略

曾几何时，我们采用各种各样的几何变换，颜色变换策略来进行数据增强。随机裁剪，颜色扰动，都对提升模型的泛化能力起着至关重要的作用。

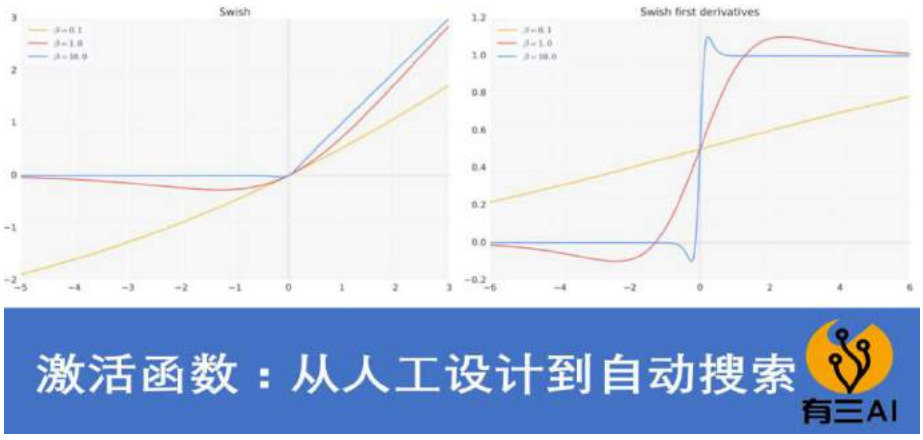
而如今，是时候寻找更好的方法了。以 Google Brain 提出的 AutoAugment 为代表的方法，**使用增强学习对不同的任务学习到了各自最合适的增强方法**，可以参考往期文章(点击图片)。



4.3、自动优化参数选择

曾几何时，我们设计，比较，分析 sigmoid，tanh，relu 等激活函数对网络性能的影响。

而 Google Brain 提出的以 Swish 为代表的方法，在一系列一元函数和二元函数组成的搜索空间中，进行了组合搜索实验，**利用数据学习到了比 ReLU 更好的激活函数**，可以参考往期文章(点击图片)。



曾几何时，我们还在争论是最大池化好还是平均池化好，如今基于数据的池化策略已经被广泛研究。

曾几何时，我们还在不知道选择什么样的归一化方法好，如今，基于数据的归一化策略也在被研究。

曾几何时，我们还在不知道选择什么样的优化方法好，如今，基于数据的优化方法也在被研究。

这些内容，可以参考公众号的《AI 初识境》，后面我们也会做更多详细的解读。

可以说，从模型的结构设计，模型的优化参数选择，数据的使用策略，深度学习正在全面走向自动化。

5、总结

很久以前，我们只会使用抽象好的数据。后来，我们学会了从数据中自己抽象特征。后来，我们发明了一个系统让它去抽象特征。再到后来，我们想让数据把系统也学了。

【AI 不惑境】网络深度对深度学习模型性能有什么影响？

本文是专栏**《AI 不惑境》**的第二篇文章，讲述模型深度与模型性能的关系。

进入到不惑境界，就是向高手迈进的开始了，在这个境界需要自己独立思考。如果说学习是一个从模仿，到追随，到创造的过程，那么到这个阶段，应该跃过了模仿和追随的阶段，进入了创造的阶段。从这个境界开始，讲述的问题可能不再有答案，更多的是激发大家一起来思考。

作者|言有三

深度学习模型之所以在各种任务中取得了成功，足够的网络深度起到了很关键的作用，那么是不是模型越深，性能就越好呢？

1 为什么加深可以提升性能

Bengio 和 LeCun 在 2017 年的文章[1]中有这么一句话，"We claim that most functions that can be represented compactly by deep architectures cannot be represented by a compact shallow architecture"，大体意思就是大多数函数如果用一个深层结构刚刚好解决问题，那么就不可能用一个更浅的同样紧凑的结构来解决。

要解决比较复杂的问题，要么增加深度，要么增加宽度，而增加宽度的代价往往远高于深度。

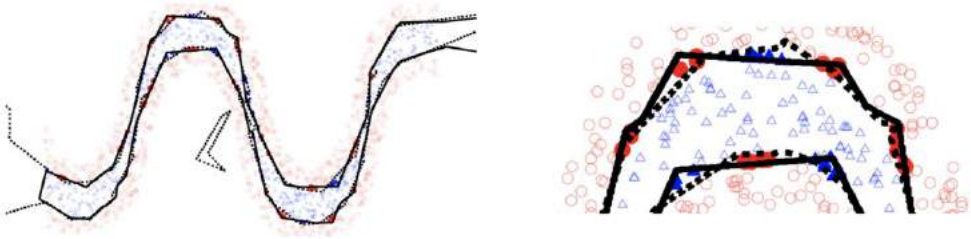
Ronen Eldan 等人甚至设计了一个能被小的 3 层网络表示，而不能被任意的 2 层网络表示的函数。总之，一定的深度是必要的。

那么随着模型的加深，到底有哪些好处呢？

1.1、更好拟合特征。

现在的深度学习网络结构的基本模块是卷积，池化，激活，这是一个标准的非线性变换模块。**更深的模型，意味着更好的非线性表达能力，可以学习更加复杂的变换，从而可以拟合更加复杂的特征输入。**

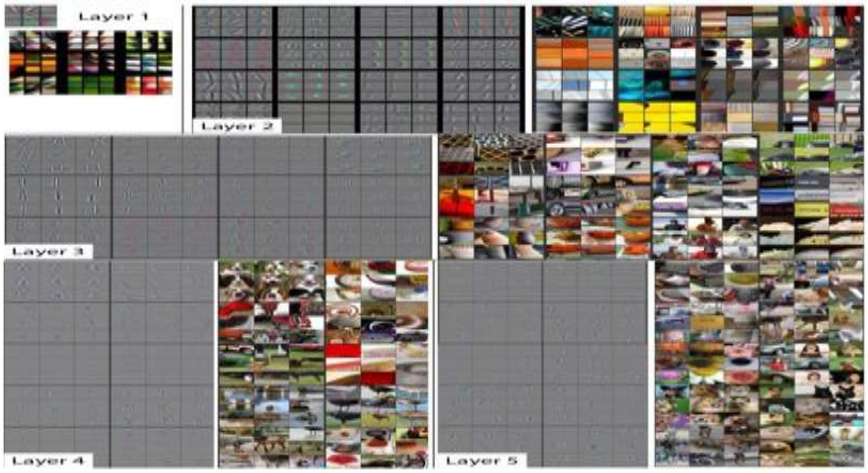
看下面的一个对比图[2]，实线是一个只有一层，20 个神经元的模型，虚线是一个 2 层，每一层 10 个神经元的模型。从图中可以看出，2 层的网络有更好的拟合能力，这个特性也适用于更深的网络。



1.2、网络更深，每一层要做的事情也更加简单了。

每一个网络层各司其职，我们从 zfnet 反卷积看一个经典的网络各个网络层学习到的权重。

第一层学习到了边缘，第二层学习到了简单的形状，第三层开始学习到了目标的形状，更深的网络层能学习到更加复杂的表达。如果只有一层，那就意味着要学习的变换非常的复杂，这很难做到。



上面就是网络加深带来的两个主要好处，**更强大的表达能力和逐层的特征学习**。

2 如何定量评估深度与模型性能

理论上一个 2 层的网络可以拟合任何有界的连续函数，但是需要的宽度很大，这在实际使用中不现实，因此我们才会使用深层网络。

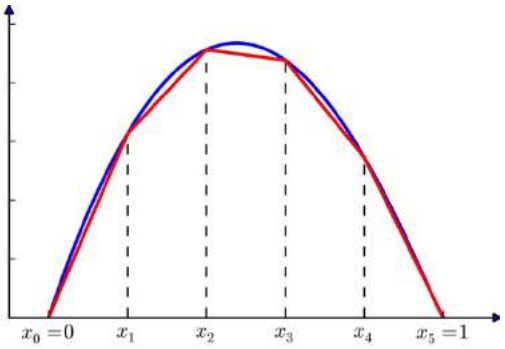
我们知道一个模型越深越好，但是怎么用一个指标直接定量衡量模型的能力和深度之间的关系，就有了**直接法和间接法两种方案**。

直接法便是定义指标理论分析网络的能力，间接法便是通过在任务中的一系列指标比如准确率等来进行比较。

2.1、直接法

早期对浅层网络的研究，通过研究函数的逼近能力，网络的 VC 维度等进行评估，但是并不适用于深层网络。

目前直接评估网络性能一个比较好的研究思路是线性区间(linear regions)。可以将神经网络的表达看作是一个分段线性函数，如果要完美的拟合一个曲线，就需要无数多的线性区间(linear regions)。线性区间越多，说明网络越灵活。



Yoshua Bengio 等人就通过**线性区间的数量来衡量模型的灵活性。一个更深的网络，可以将输入空间分为更多的线性响应空间，它的能力是浅层网络的指数级倍。**

对于一个拥有 n0 个输入，n 个输出，kn 个隐藏层的单层网络，其最大数量为：

$$O\left(k^{n_0-1}n^{n_0-1}\right)$$

对于拥有同样多的参数，n0 个输入，n 个输出，k 个隐藏层，每一层 n 个节点的多层网络，其最大数量为：

$$\Omega\left(\left\lfloor\frac{n}{n_0}\right\rfloor^{k-1}\frac{n^{n_0-2}}{k}\right)$$

因为 n0 通常很小，所以多层网络的数量是单层的指数倍(体现在 k 上)，计算是通过计算几何学来完成，大家可以参考论文[3]。

除此之外还有一些其他的研究思路，比如 monica binachini[4]等使用的 betti number，Maithra Raghu 等提出的 trajectory length[5]。

虽然在工程实践中这些指标没有多少意义甚至不一定有效，但是为我们理解深度和模型性能的关系提供了理论指导。

2.2、间接法

间接法就是展现实验结果了，网络的加深可以提升模型的性能，这几乎在所有的经典网络上都可以印证。比较不同的模型可能不够公平，那就从同一个系列的模型来再次感受一下，看看 VGG 系列模型，ResNet 系列模型，结果都是从论文中获取。

ResNet 10 crop(256crop224)

网络	Top-1 error	Top-5 error
Res34	24.19	7.4
Res50	22.85	6.71
Res101	21.75	6.05
Res152	21.43	5.71

VGG single crop(256crop224)

网络	Top-1 error	Top-5 error
VGG11	29.6	10.4
VGG13	28.7	9.9
VGG16	27.0	8.8
VGG19	27.3	9.0

在一定的范围内，网络越深，性能的确就越好。

3 加深就一定更好吗？

前面说到加深在一定程度上可以提升模型性能，但是未必就是网络越深就越好，我们从**性能提升和优化两个方面**来看。

3.1、加深带来的优化问题

ResNet 为什么这么成功，就是因为它使得深层神经网络的训练成为可行。虽然好的初始化，BN 层等技术也有助于更深层网络的训练，但是很少能突破 30 层。

VGGNet19 层，GoogleNet22 层，MobileNet28 层，经典的网络超过 30 层的也就是 ResNet 系列常见的 ResNet50，ResNet152 了。虽然这跟后面 ImageNet 比赛的落幕，大家开始追求更加高效实用的模型有关系，另一方面也是训练的问题。

深层网络带来的**梯度不稳定，网络退化**的问题始终都是存在的，可以缓解，没法消除。这就有可能出现网络加深，性能反而开始下降。

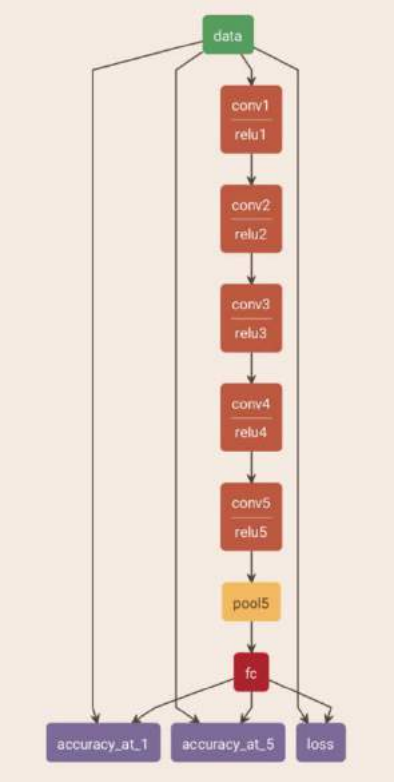
3.2、网络加深带来的饱和

网络的深度不是越深越好，下面我们通过几个实验来证明就是了。公开论文中使用的 ImageNet 等数据集研究者已经做过很多实验了，我们另外选了两个数据集和两个模型。

第一个数据集是 GHIM 数据集，第二个数据集是从 Place20 中选择了 20 个类别，可见两者一个比较简单，一个比较困难。

第一个模型就是简单的卷积+激活的模型，第二个就是 mobilenet 模型。

首先我们看一下第一个模型的基准结构，包含 5 层卷积和一个全连接层， 因此我们称其为 allconv6 吧，表示深度为 6 的一个卷积网络。

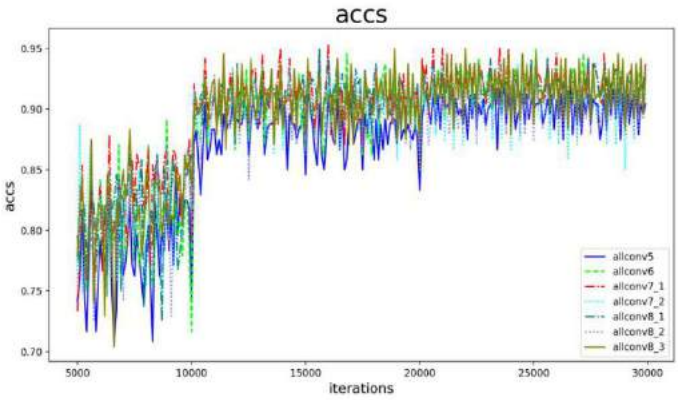


接下来我们试验各种配置，从深度为 5 到深度为 8，下面是每一个网络层的 stride 和通道数的配置。

网络	conv1	conv2	conv3	conv4	conv5	conv6	conv7
Allconv5	4	2	2	2	-	-	-
Allconv6	2	2	2	2	2	-	-
Allconv7_1	1	2	2	2	2	2	-
Allconv7_2	2	2	2	2	2	1	
Allconv8_1	1	1	2	2	2	2	2
Allconv8_2	1	2	2	2	2	2	1
Allconv8_3	2	2	2	2	2	1	1

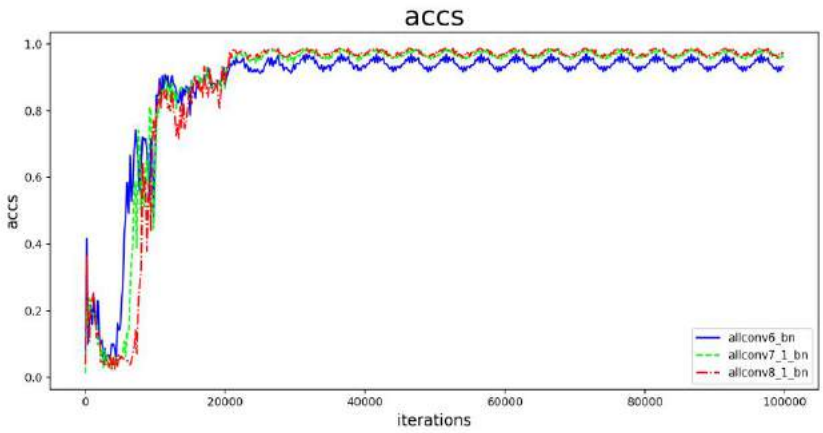
网络	conv1	conv2	conv3	conv4	conv5	conv6	conv7
Allconv5	64	64	128	128	-	-	-
Allconv6	64	64	128	128	256	-	-
Allconv7_1	64	64	128	128	256	256	-
Allconv7_2	64	64	128	128	256	256	
Allconv8_1	64	64	128	128	256	256	256
Allconv8_2	64	64	128	128	256	256	256
Allconv8_3	64	64	128	128	256	256	256

我们看结果，优化都是采用了同一套参数配置，而且经过了调优，具体细节篇幅问题就不多说了。

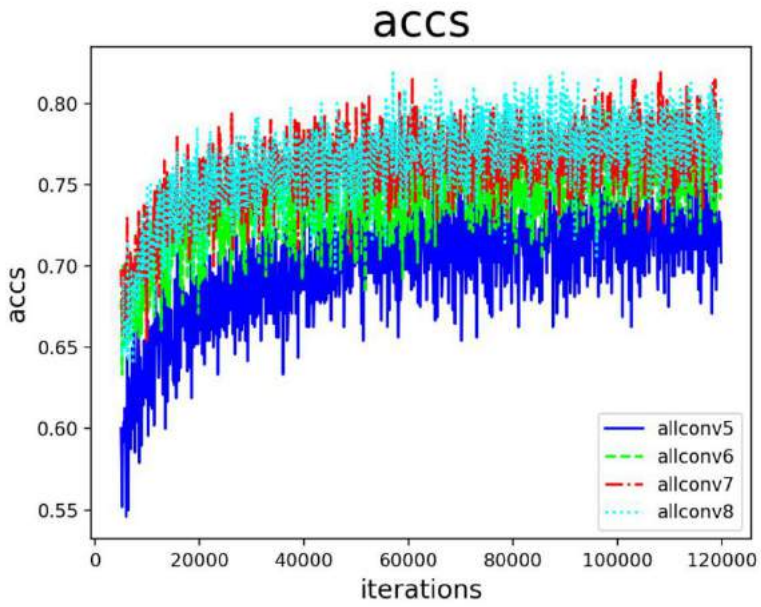


看的出来网络加深性能并未下降，但是也没有多少提升了。allconv5 的性能明显更差，深度肯定是其中的一个因素。

我们还可以给所有的卷积层后添加 BN 层做个试验，结果如下，从 allconv7_1 和 allconv8_1 的性能相当且明显优于 allconv6 可以得出与刚才同样的结论。



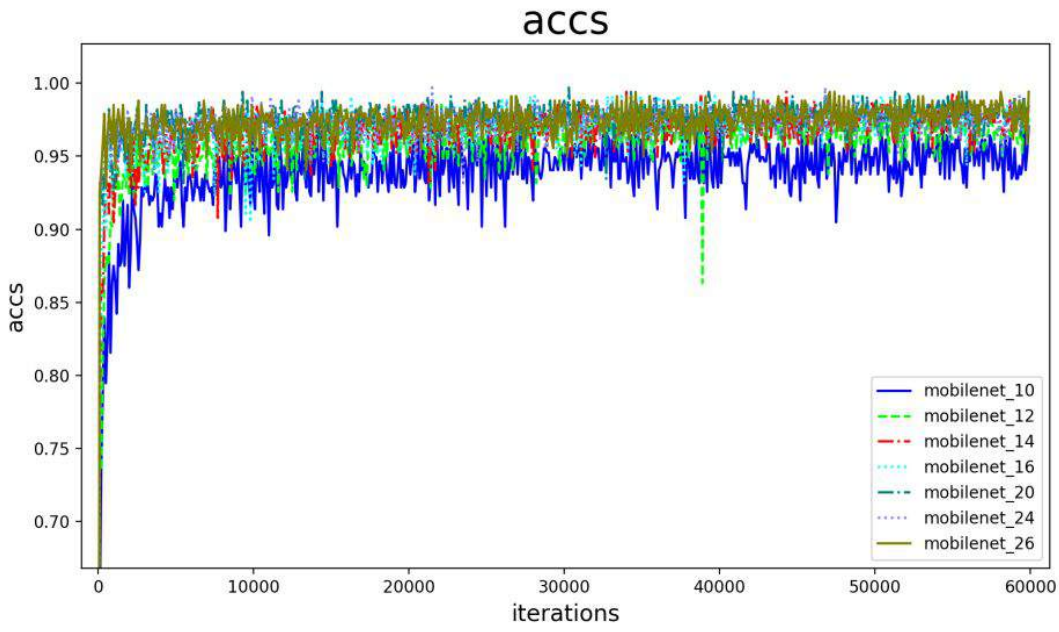
那么，对于更加复杂的数据集，表现又是如何呢？下面看在 place20 上的结果，更加清晰了。



allconv5，allconv6 结果明显比 allconv7，allconv8 差，而 allconv7 和 allconv8 性能相当。所以从 allconv 这个系列的网络结构来看，随着深度增加到 allconv7，之后再简单增加深度就难以提升了。

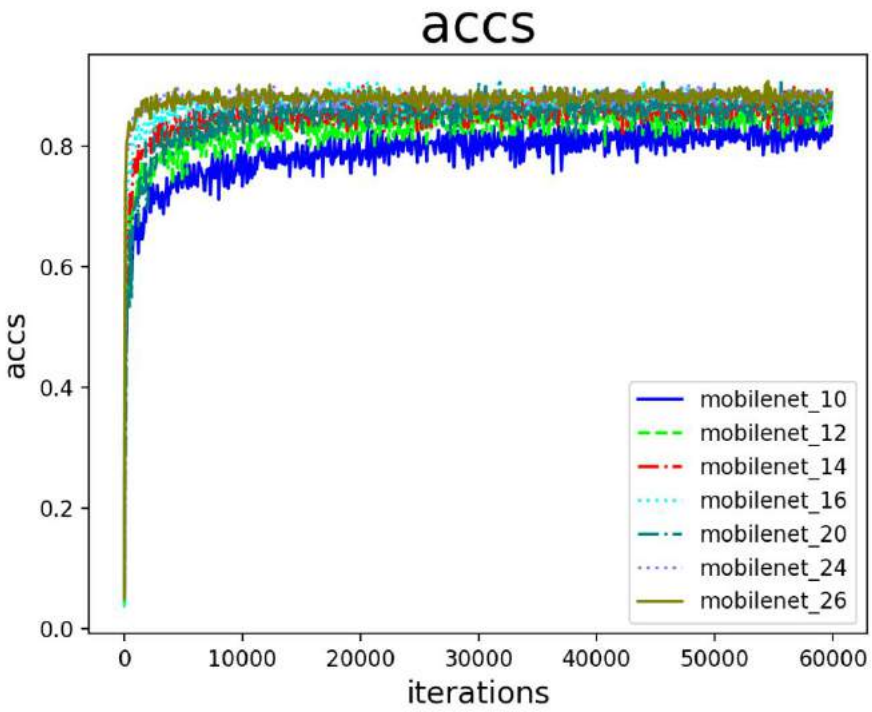
接下来我们再看一下**不同深度的 mobilenet 在这两个数据集上的表现**，原始的 mobilenet 是 28 层的结构。

不同深度的 MobileNet 在 GHIM 数据集的结果如下：



看得出来当模型到 16 层左右后，基本就饱和了。

不同深度的 MobileNet 在 Place20 数据集的结果如下：



与 GHIM 的结果相比，深度带来的提升更加明显一些，不过也渐趋饱和。

这是必然存在的问题，哪有一直加深一直提升的道理，只是如何去把握这个深度，尚且无法定论，只能依靠更多的实验了。

除此之外，模型加深还可能出现的一些问题是**导致某些浅层的学习能力下降**，限制了深层网络的学习，这也是跳层连接等结构能够发挥作用的很重要的因素。

关于网络深度对模型性能的影响，这次就先说这么多，更多就去听我的知乎 live 吧。

[1] Bengio Y, LeCun Y. Scaling learning algorithms towards AI[J]. Large-scale kernel machines, 2007, 34(5): 1-41.

[2] Montufar G F, Pascanu R, Cho K, et al. On the number of linear regions of deep neural networks[C]//Advances in neural information processing systems. 2014: 2924-2932.

[3] Pascanu R, Montufar G, Bengio Y. On the number of response regions of deep feed forward networks with piece-wise linear activations[J]. arXiv preprint arXiv:1312.6098, 2013.

[4] Bianchini M, Scarselli F. On the complexity of neural network classifiers: A comparison between shallow and deep architectures[J]. IEEE transactions on neural networks and learning systems, 2014, 25(8): 1553-1565.

[5] Raghu M, Poole B, Kleinberg J, et al. On the expressive power of deep neural networks[C]//Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017: 2847-2854.

4 总结

深度学习的名字中就带着“深”，可见深度对模型的重要性。这一次我们讲述了深度对模型带来提升的原理，如何定量地评估深度对模型性能的贡献，以及加深网络会遇到的问题。

【AI 不惑境】网络的宽度如何影响深度学习模型的性能？

本文是专栏**《AI 不惑境》**的第三篇文章，讲述模型宽度与模型性能的关系。进入到不惑境界，就是向高手迈进的开始了，在这个境界需要自己独立思考。如果说学习是一个从模仿，到追随，到创造的过程，那么到这个阶段，应该跃过了模仿和追随的阶段，进入了创造的阶段。从这个境界开始，讲述的问题可能不再有答案，更多的是激发大家一起来思考。

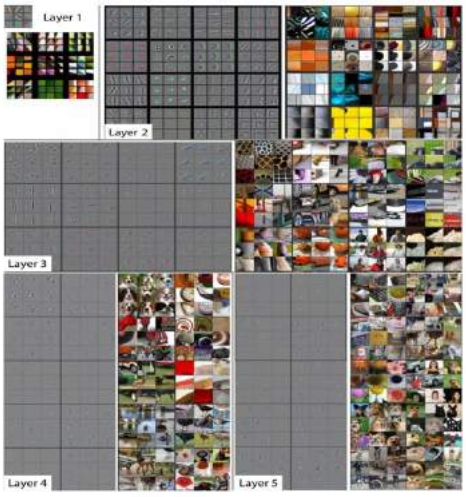
作者|言有三

上一篇咱们说到深度学习模型之所以在各种任务中取得了成功，足够的网络深度起到了很关键的作用。

在一定的程度上，网络越深，性能越好。这一次我们来考虑另一个维度，**宽度，即通道(channel)的数量**。注意我们这里说的和宽度学习一类的模型没有关系，而是特指深度卷积神经网络的宽度。

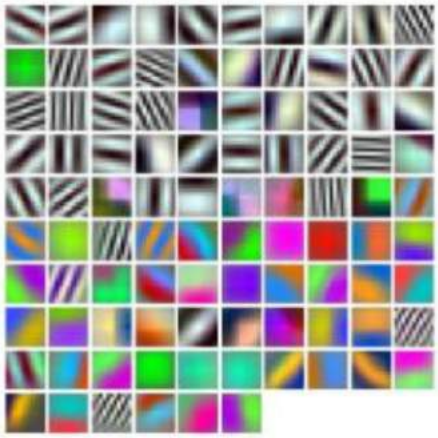
1 为什么需要足够的宽度

网络更深带来的一个非常大的好处，就是逐层的抽象，不断精炼提取知识，如下图第一层学习到了边缘，第二层学习到了简单的形状，第三层开始学习到了目标的形状，更深的网络层能学习到更加复杂的表达。如果只有一层，那就意味着要学习的变换非常的复杂，这很难做到。



而宽度就起到了另外一个作用，那就是让每一层学习到更加丰富的特征，比如不同方向，不同频率的纹理特征。

下面是 AlexNet 模型的第一个卷积层的 96 个通道，尽管其中有一些形状和纹理相似的卷积核(这将成为优化宽度的关键)，还是可以看到各种各种的模式。



因为该卷积层的输入是 RGB 彩色图，所以这里就将其可视化为 3 通道的彩色图，每一个大小是 11*11。

有的是彩色有的是灰色，说明有的侧重于提取纹理信息，有的侧重于提取颜色信息。

可以发现卷积核可视化之后和 Gabor 特征算子其实很像。Gabor 特征算子就是使用一系列不同频率的 Gabor 滤波核与图像卷积，得到图像上的每个点和附近区域的频率分布。通常有 8 个方向，5 个尺度。

太窄的网络，每一层能捕获的模式有限，此时网络再深都不可能提取到足够的信息往下层传递。

2 网络到底需要多宽

那么一个网络是越宽越好吗？我们又该如何利用好宽度呢？

2.1、网络宽度的下限在哪？

就算一个网络越宽越好，我们也希望效率越高越好，因为宽度带来的计算量是成平方数增长的。我们知道对于一个模型来说，浅层的特征非常重要，因此网络浅层的宽度是一个非常敏感的系数，那么发展了这么久，那些经典的网络第一个卷积层的宽度都是多少呢？

网络	通道数	发表时间
AlexNet	96	2012
GoogLeNet	64	2014
vgg16	64	2014
ResNet18	64	2015
SqueezeNet1.0	64	2017
DenseNet121	64	2017
MobileNet1.0	32	2017
ShuffleNet1.0	24	2017

从 AlexNet 的 96 层到 Vgg，Resnet 等多数网络使用的 64 层，到高效网络 Mobilenet 的 32 层和 Shufflenet 的 24 层，似乎已经探到了下限，再往下性能就无法通过其他的方法来弥补了。

前次我们说过有许多的研究都验证了网络必须具有足够的深度才能逼近一些函数，比如文[1]中构造的 3 层网络，如果想要 2 层网络能够逼近表达能力，宽度会是指数级的增加。

那么反过来，是不是也有一些函数只有足够宽才能够表达呢？

针对网络宽度的研究虽不如网络深度多，但是也有学者做了相关研究。文[2]中就提出了任何 Lebesgue-integrable 函数，不能被一个宽度小于 n 的 ReLU 网络逼近，n 是输入的维度，Lebesgue-integrable 函数就是满足下面积分条件的函数。

$$\int_{\mathbb{R}^n} |f(x)| \mathrm{d}x < \infty$$

不过与深度不同的是，这样的一些函数宽度减少后，用于补偿模型性能的深度不是呈指数级增长，而是多项式增长，这似乎反应了宽度并没有深度那么重要。

不过不管怎么样，当前研究者们都从理论上探索了宽度和深度的下限，表明宽度和深度是缺一不可的。

2.2、网络宽度对模型性能的影响

网络的宽度自然也不是越宽越好，下面我们看看网络的宽度带来的性能提升。

我们看一下 Mobilenet 网络的结果，Mobilenet 研究了网络的宽度对性能的影响，通过一个乘因子来对每一层的宽度进行缩放，它们试验了 1，0.75，0.5 和 0.25 共 4 个值。

Table 6. MobileNet Width Multiplier

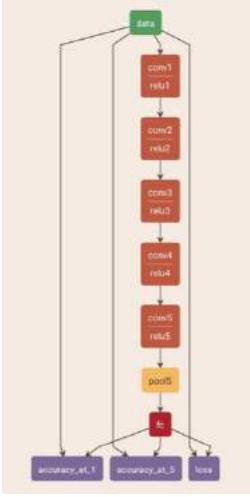
Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

从上面结果可以看得出来，性能是持续下降的。

那么，是不是网络越宽越好呢？下面我们还是通过几个实验来证明就是了。公开论文中使用的 ImageNet 等数据集研究者已经做过很多实验了，我们另外选了两个数据集和一个全卷积模型。

第一个数据集是 GHIM 数据集，第二个数据集是从 Place20 中选择了 20 个类别，可见两者一个比较简单，一个比较困难。

使用全卷积模型的基准结构，包含 5 层卷积和一个全连接层， 因此我们称其为allconv6吧，表示深度为 6 的一个卷积网络。



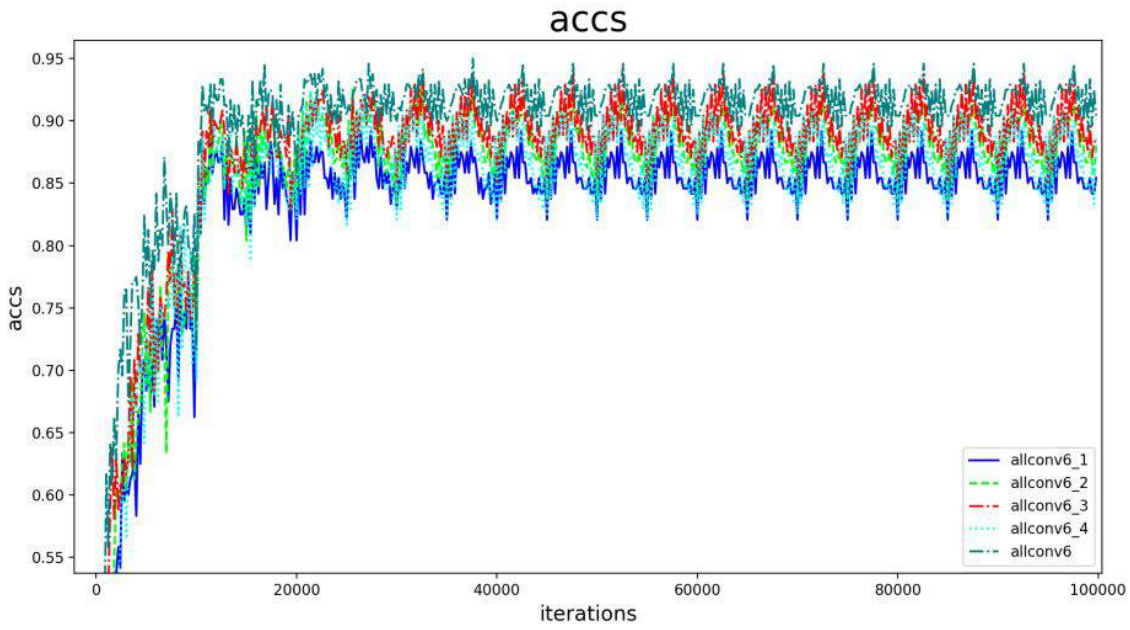
对这个网络的各个卷积层，我们也设置了不同的参数配置如下，每一个卷积层的 stride 都等于 2。

网络配置	Conv1	Conv2	Conv3	Conv4	Conv5
Baseline	64	64	128	128	256
Allconv6_1	16	16	32	32	64
Allconv6_2	32	32	32	32	64
Allconv6_3	16	16	64	64	64
Allconv6_4	16	16	32	32	128
Allconv6_5	32	32	64	64	128
Allconv6_6	64	64	64	64	128
Allconv6_7	32	32	128	128	128
Allconv6_8	32	32	64	64	256

首先我们比较 Allconv6_1，Allconv6_2，Allconv6_3，Allconv6_4 这 4 个模型和基准模型的结果，它们是以 Allconv6_1 为基础的模型。

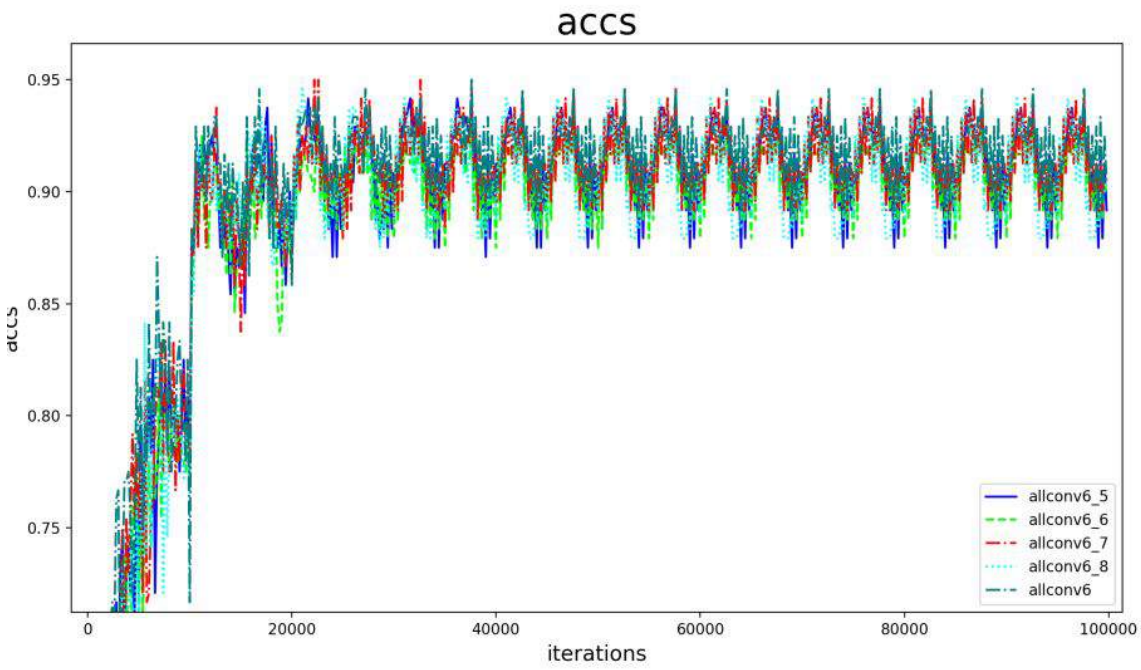
Allconv6_1 是各个通道数为 baseline 的四分之一的网络，而 Allconv6_2，Allconv6_3，Allconv6_4 分别是将 Allconv6_1 的第 1，2 层，第 3，4 层，第 5 层卷积通道数加倍的网络。

在 GHIM 数据集上的收敛结果如下：

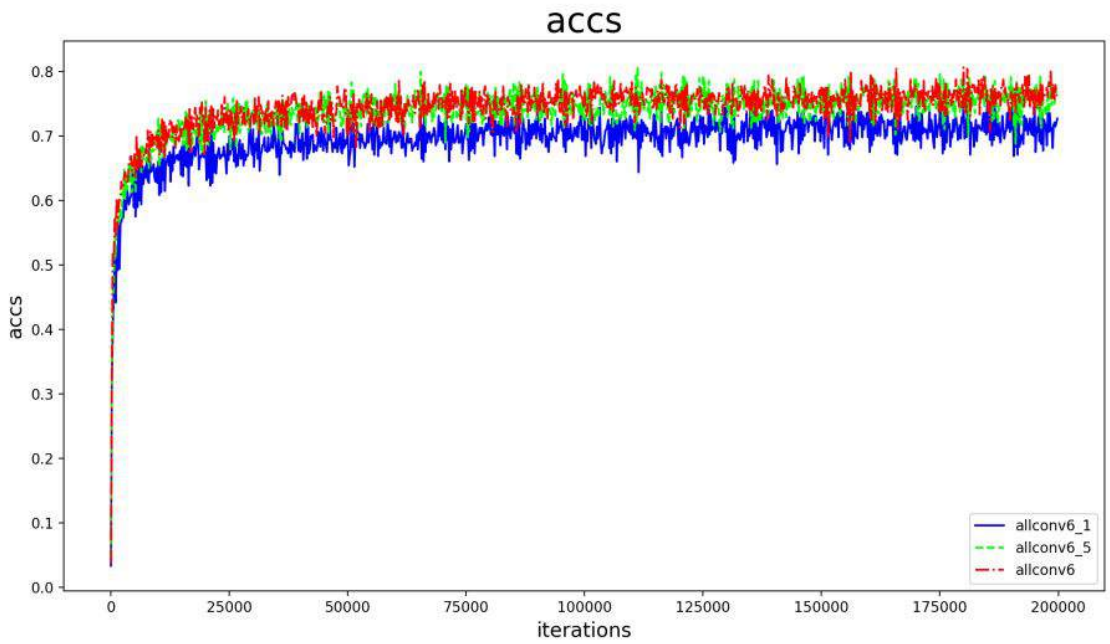


从上图结果可以看出，基准模型 allconv6 的性能最好，allconv6_2，allconv6_3，allconv6_4 的模型性能都超过 allconv6_1，说明此时增加任何一个网络层的通道数都有益于模型性能的提升，而且性能仍旧未超过基准模型。

然后我们再看 allconv6_5，allconv6_6，allconv6_7，allconv6_8 与基准模型的对比，**allconv6_5 的各层的通道数只有 baseline 模型的一半**。



从上图可以看出，模型的性能相差不大，这说明 allconv6_5 已经有足够好的宽度，再增加无益于性能的提升。这一点可以通过 Place20 上的实验结果进行证明，结果如下：



2.3、网络宽度和深度谁更加重要？

这个问题目前没有答案，两者都很重要，不过目前的研究是模型性能对深度更加敏感，而调整宽度更加有利于提升模型性能。

Mobilenet 的作者们将深层更窄的网络和浅层更宽的网络进行了对比，去掉了 conv5_2 到 conv5_6 这 5 层不改变分辨率的 depth seperable 卷积块，结果对比如下：

Table 5. Narrow vs Shallow MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.75 MobileNet	68.4%	325	2.6
Shallow MobileNet	65.3%	307	2.9

更窄的网络拥有了更少的参数和更好的性能，这似乎也验证了增加网络的深度比增加网络的宽度更有利于提升性能。

在 Wide Resnet 网络中，作者们在 CIFAR10 和 CIFAR100 上用参数只是稍微增加的一个 16 层的宽网络取得了比 1000 层的窄网络更好的性能，而且计算代价更低。在 ImageNet 上 50 层的宽 Resnet 在参数增加少量的基础上，也比相应的 ResNet152 层的性能更好。

Model	top-1 err, %	top-5 err, %	#params	time/batch 16
ResNet-50	24.01	7.02	25.6M	49
ResNet-101	22.44	6.21	44.5M	82
ResNet-152	22.16	6.16	60.2M	115
WRN-50-2-bottleneck	21.9	6.03	68.9M	93
pre-ResNet-200	21.66	5.79	64.7M	154

另一方面，宽度相对于深度对 GPU 也更加友好，因为 GPU 是并行处理的，许多研究也表明加宽网络比加深网络也更加容易训练。

没有谁更重要，根据笔者的经验，我们应该优先调整网络的宽度。

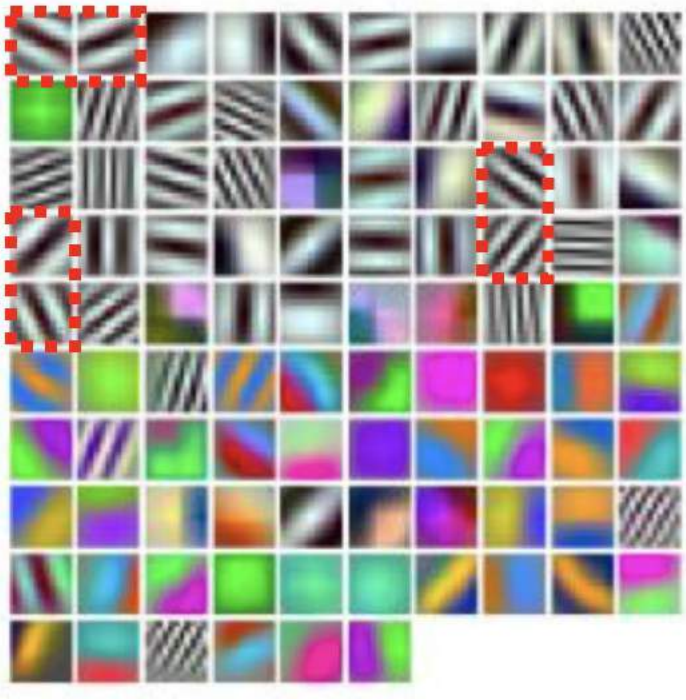
3 如何更加有效地利用宽度？

从前面的结果我们可知，网络的宽度是非常关键的参数，它体现在两个方面：(1) 宽度对计算量的贡献非常大。(2)宽度对性能的影响非常大。

我们的追求当然是越窄同时性能越高的网络，确实很贪婪，不过这是要实现的目标，可以从以下几个方向入手。

3.1、提高每一层通道的利用率

宽度既然这么重要，那么每一个通道就要好好利用起来，所以，第一个发力点，便是提高每一层的通道利用率。下面我们首先观察一下 AlexNet 网络的第一个卷积层。

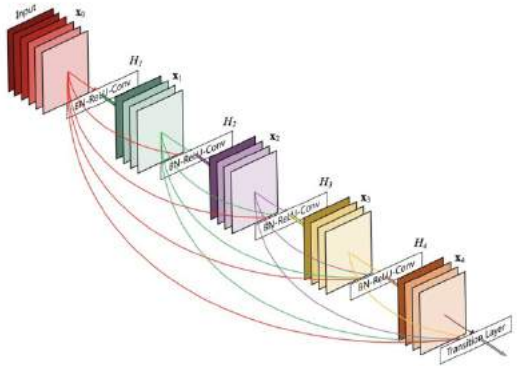


看出来了吧，有些卷积核很相似，相互之间可以通过反转得到，比如前面两个，那么就只需要学习一个就行了，这便是网络参数互补现象，如果将减半后的通道补上它的反，会基本上相当于原有的模型。

基于这个原理，文[3]便是通过输入通道取反和输入通道进行 concat 的方式来扩充通道。这样仅仅以原来一半的计算量便维持了原来的网络宽度和性能。

3.2、用其他通道的信息来补偿

这个思想在 DenseNet[4]网络中被发挥地淋漓尽致。DenseNet 网络通过各层之间进行 concat，可以在输入层保持非常小的通道数的配置下，实现高性能的网络。



这一次的网络宽度对模型性能的影响就说到这里，更多请大家至我的知乎 live 中交流。

参考文献

[1] Eldan R, Shamir O. The power of depth for feedforward neural networks[C]//Conference on learning theory. 2016: 907-940.
[2] Lu Z, Pu H, Wang F, et al. The expressive power of neural networks: A view from the width[C]//Advances in Neural Information Processing Systems. 2017: 6231-6239.
[3] Shang W, Sohn K, Almeida D, et al. Understanding and improving convolutional neural networks via concatenated rectified linear units[C]//international conference on machine learning. 2016: 2217-2225.
[4] Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 4700-4708.

4 总结

深度学习成功的关键在于深，但是我们也不能忘了它的宽度，即通道数目，这对于模型性能的影响不亚于深度，在计算量上的影响甚至尤比深度更加重要。

【AI 不惑境】学习率和 batchsize 如何影响模型的性能？

本文是专栏**《AI 不惑境》**的第四篇文章，讲述学习率以及 batchsize 与模型性能的关系。进入到不惑境界，就是向高手迈进的开始了，在这个境界需要自己独立思考。如果说学习是一个从模仿，到追随，到创造的过程，那么到这个阶段，应该跃过了模仿和追随的阶段，进入了创造的阶段。从这个境界开始，讲述的问题可能不再有答案，更多的是激发大家一起来思考。

作者|言有三

前几期我们讲述了数据，模型的深度，宽度对深度学习模型性能的影响，这一次我们讲述**学习率和 batchsize 对模型性能的影响，在实践中这两个参数往往一起调整**。

1 为什么说学习率和 batchsize

目前深度学习模型多采用批量随机梯度下降算法进行优化，随机梯度下降算法的原理如下，

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t).$$

n 是批量大小(batchsize)， η是学习率(learning rate)。可知道除了梯度本身，**这两个因子直接决定了模型的权重更新**，从优化本身来看它们是影响模型性能收敛最重要的参数。

学习率直接影响模型的收敛状态，batchsize 则影响模型的泛化性能，两者又是分子分母的直接关系，相互也可影响，因此这一次来详述它们对模型性能的影响。

2 学习率如何影响模型性能？

通常我们都需要合适的学习率才能进行学习，要达到一个强的凸函数的最小值，学习率的调整应该满足下面的条件，i 代表第 i 次更新。

$$\sum_{i=1}^{\infty} \epsilon_i = \infty,$$

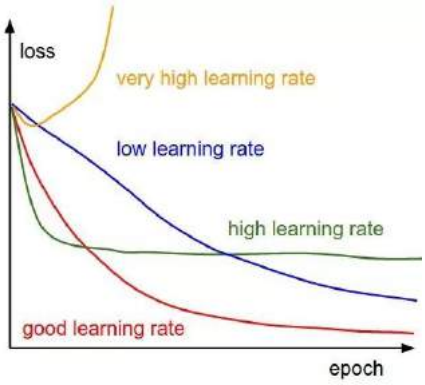
$$\sum_{i=1}^{\infty} \epsilon_i^2 < \infty.$$

第一个式子决定了不管初始状态离最优状态多远，总是可以收敛。第二个式子约束了学习率随着训练进行有效地降低，保证收敛稳定性，各种自适应学习率算法本质上就是不断在调整各个时刻的学习率。

学习率决定了权重迭代的步长，因此是一个非常敏感的参数，它对模型性能的影响体现在两个方面，**第一个是初始学习率的大小，第二个是学习率的变换方案。**

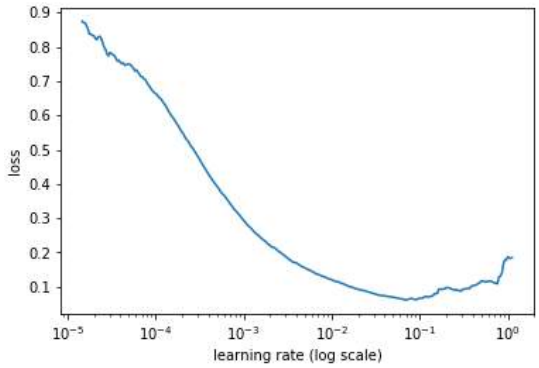
2.1、初始学习率大小对模型性能的影响

初始的学习率肯定是有一个最优值的，过大则导致模型不收敛，过小则导致模型收敛特别慢或者无法学习，下图展示了不同大小的学习率下模型收敛情况的可能性，图来自于 cs231n。



那么在不考虑具体的优化方法的差异的情况下，怎样确定最佳的初始学习率呢？

通常可以采用最简单的搜索法，即从小到大开始训练模型，然后记录损失的变化，通常会记录到这样的曲线。



随着学习率的增加，损失会慢慢变小，而后增加，而最佳的学习率就可以从其中损失最小的区域选择。

有经验的工程人员常常根据自己的经验进行选择，比如 0.1，0.01 等。

随着学习率的增加，模型也可能会从欠拟合过度到过拟合状态，在大型数据集上的表现尤其明显，笔者之前在 Place365 上使用 DPN92 层的模型进行过实验。随着学习率的增强，模型的训练精度增加，直到超过验证集。

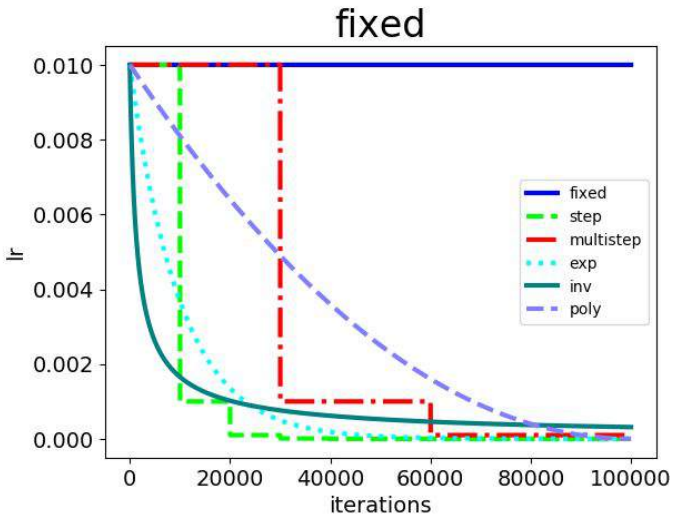
lr	Train acc	Val acc
0.001	0.84	0.91
0.005	0.91	0.95
0.01	0.93	0.95
0.015	0.95	0.947

2.2、学习率变换策略对模型性能的影响

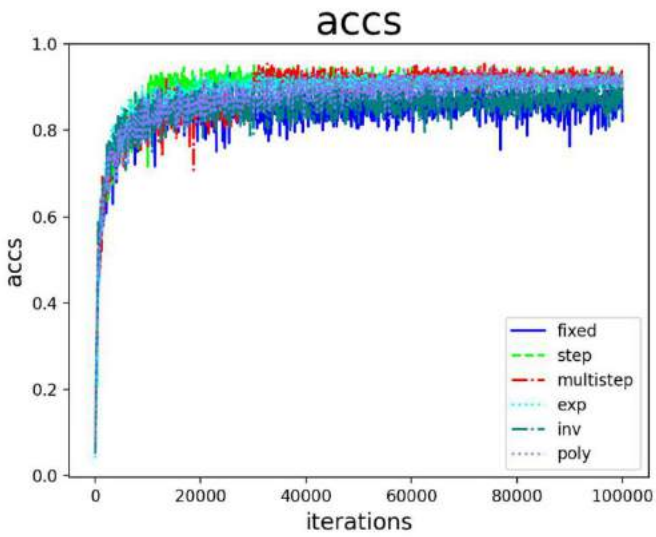
学习率在模型的训练过程中很少有不变的，通常会有两种方式对学习率进行更改，一种是**预设规则学习率变化法**，一种是**自适应学习率变换方法**。

2.2.1 预设规则学习率变化法

常见的策略包括 fixed，step，exp，inv，multistep，poly，sigmoid 等，集中展示如下：



笔者之前做过一个实验来观察在 SGD 算法下，各种学习率变更策略对模型性能的影响，具体的结果如下：



从结果来看：

step，multistep方法的收敛效果最好，这也是我们平常用它们最多的原因。虽然学习率的变化是最离散的，但是并不影响模型收敛到比较好的结果。

其次是exp，poly。它们能取得与step，multistep相当的结果，也是因为学习率以比较好的速率下降，虽然变化更加平滑，但是结果也未必能胜过 step 和 multistep 方法，在这很多的研究中都得到过验证，离散的学习率变更策略不影响模型的学习。

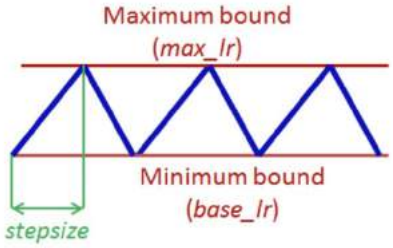
inv 和 fixed 的收敛结果最差。这是比较好解释的，因为 fixed 方法始终使用了较大的学习率，而 inv 方法的学习率下降过程太快。

关于以上内容的完整分析结果，可以查看往期文章：

【模型训练】如何选择最适合你的学习率变更策略

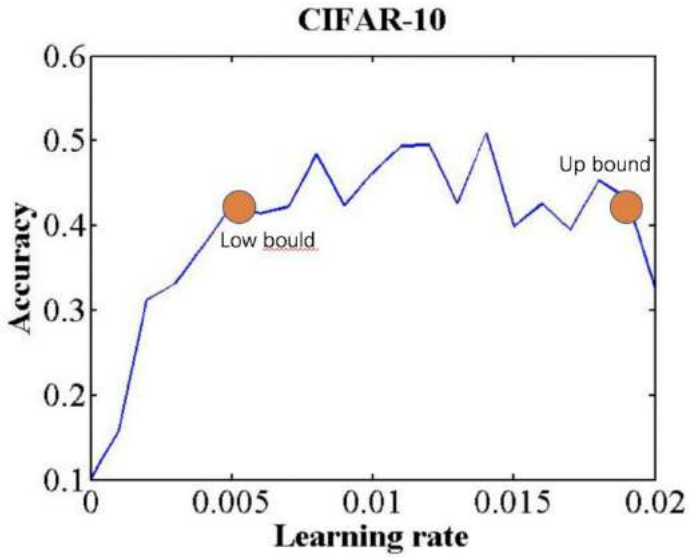
从上面的结果可以看出，对于采用非自适应学习率变换的方法，学习率的绝对值对模型的性能有较大影响，研究者常使用 step 变化策略。

目前学术界也在探索一些最新的研究方法，比如 cyclical learning rate，示意图如下：

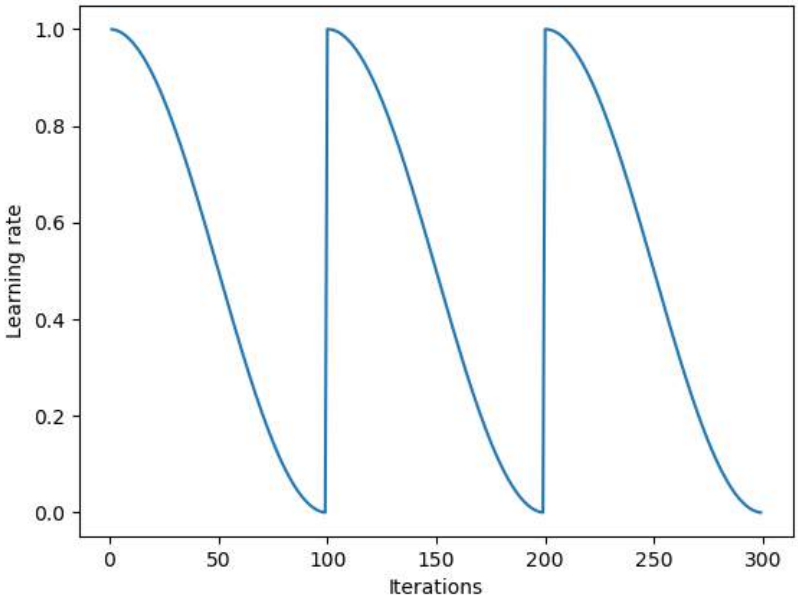


实验证明通过设置上下界，让学习率在其中进行变化，可以在模型迭代的后期更有利于克服因为学习率不够而无法跳出鞍点的情况。

确定学习率上下界的方法则可以使用 LR range test 方法，即使用不同的学习率得到精度曲线，然后获得精度升高和下降的两个拐点，或者将精度最高点设置为上界，下界设置为它的 1/3 大小。



SGDR 方法则是比 cyclical learning rate 变换更加平缓的周期性变化方法，如下图，效果与 cyclical learning rate 类似。

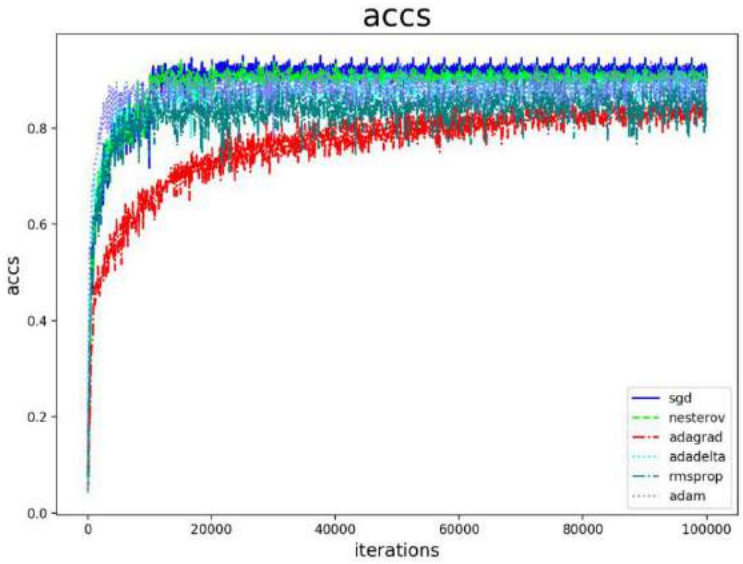


2.2.2 自适应学习率变化法

自适应学习率策略以 Adagrad，Adam 等为代表，我们在公众号已经说得非常多了，这里就不再做原理上的讲述，可以查看往期介绍：

【AI 初识境】为了围剿 SGD 大家这些年想过的那十几招

原理上各种改进的自适应学习率算法都比SGD算法更有利于性能的提升，但实际上精细调优过的 SGD 算法可能取得更好的结果，在很多的论文[3-4]中都得到过验证，我们在实验中也多次证明过这一点，如下图。



2.3、小结

不考虑其他任何因素，学习率的大小和迭代方法本身就是一个非常敏感的参数。如果经验不够，还是考虑从 Adam 系列方法的默认参数开始，如果经验丰富，可以尝试更多的实验配置。

3 Batchsize 如何影响模型性能？

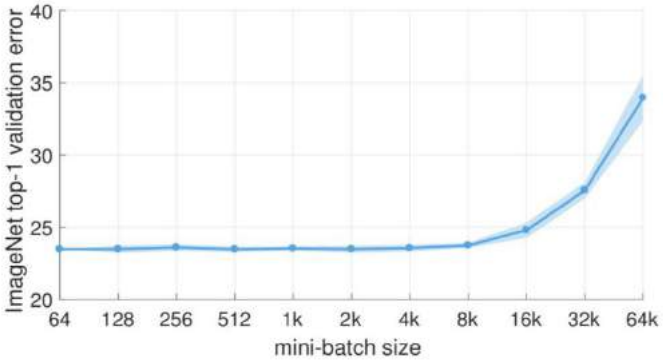
模型性能对 batchsize 虽然没有学习率那么敏感，但是在进一步提升模型性能时，batchsize 就会成为一个非常关键的参数。

3.1 大的 batchsize 减少训练时间，提高稳定性

这是肯定的，同样的 epoch 数目，大的 batchsize 需要的 batch 数目减少了，所以可以减少训练时间，目前已经有多篇公开论文在 1 小时内训练完 ImageNet 数据集。另一方面，大的 batch size 梯度的计算更加稳定，因为模型训练曲线会更加平滑。在微调的时候，大的 batch size 可能会取得更好的结果。

3.2 大的 batchsize 泛化能力下降

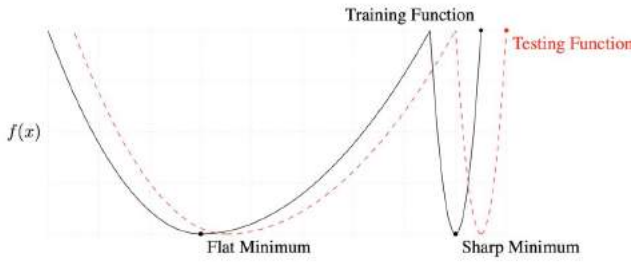
在一定范围内，增加batchsize有助于收敛的稳定性，但是随着batchsize的增加，模型的性能会下降，如下图，来自于文[5]。



这是研究者们普遍观测到的规律，虽然可以通过一些技术缓解。这个导致性能下降的 batch size 在上图就是 8000 左右。

那么这是为什么呢？

研究[6]表明大的 batchsize 收敛到 sharp minimum，而小的 batchsize 收敛到 flat minimum，后者具有更好的泛化能力。两者的区别就在于变化的趋势，一个快一个慢，如下图，造成这个现象的主要原因是小的 batchsize 带来的噪声有助于逃离 sharp mininum。



Hoffer[7]等人的研究表明，**大的 batchsize 性能下降是因为训练时间不够长，本质上并不少 batchsize 的问题**，在同样的 epochs 下的参数更新变少了，因此需要更长的迭代次数。

3.3 小结

batchsize 在变得很多时，会降低模型的泛化能力。在此之下，模型的性能变换随 batch size 通常没有学习率敏感。

4 学习率和 batchsize 的关系

通常当我们增加 batchsize 为原来的 N 倍时，要保证经过同样的样本后更新的权重相等，按照线性缩放规则，学习率应该增加为原来的 N 倍[5]。但是如果要保证权重的方差不变，则学习率应该增加为原来的 sqrt(N)倍[7]，目前这两种策略都被研究过，使用前者的明显居多。

从两种常见的调整策略来看，学习率和 batchsize 都是同时增加的。学习率是一个非常敏感的因子，不可能太大，否则模型会不收敛。同样 batchsize 也会影响模型性能，那实际使用中都如何调整这两个参数呢？

研究[8]表明，**衰减学习率可以通过增加 batchsize 来实现类似的效果**，这实际上从 SGD 的权重更新式子就可以看出来两者确实是等价的，文中通过充分的实验验证了这一点。

研究[9]表明，**对于一个固定的学习率，存在一个最优的 batchsize 能够最大化测试精度**，这个 batchsize 和学习率以及训练集的大小正相关。

对此实际上是有两个建议：

- **如果增加了学习率，那么 batch size 最好也跟着增加，这样收敛更稳定。**
- **尽量使用大的学习率，因为很多研究都表明更大的学习率有利于提高泛化能力。**如果真的要衰减，可以尝试其他办法，比如增加 batch size，学习率对模型的收敛影响真的很大，慎重调整。

关于学习率和 batch size 这次就说这么多，感兴趣可以自行拓展阅读。

参考文献

[1] Smith L N. Cyclical learning rates for training neural networks[C]//2017 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2017: 464-472.
[2] Loshchilov I, Hutter F. Sgdr: Stochastic gradient descent with warm restarts[J]. arXiv preprint arXiv:1608.03983, 2016.
[3] Reddi S J, Kale S, Kumar S. On the convergence of adam and beyond[J]. 2018.
[4] Keskar N S, Socher R. Improving generalization performance by switching from adam to sgd[J]. arXiv preprint arXiv:1712.07628, 2017.
[5] Goyal P, Dollar P, Girshick R B, et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.[J]. arXiv: Computer Vision and Pattern Recognition, 2017.
[6] Keskar N S, Mudigere D, Nocedal J, et al. On large-batch training for deep learning: Generalization gap and sharp minima[J]. arXiv preprint arXiv:1609.04836, 2016.
[7] Hoffer E, Hubara I, Soudry D. Train longer, generalize better: closing the generalization gap in large batch training of neural networks[C]//Advances in Neural Information Processing Systems. 2017: 1731-1741.
[8] Smith S L, Kindermans P J, Ying C, et al. Don't decay the learning rate, increase the batch size[J]. arXiv preprint arXiv:1711.00489, 2017.
[9] Smith S L, Le Q V. A bayesian perspective on generalization and stochastic gradient descent[J]. arXiv preprint arXiv:1710.06451, 2017.

5 总结

学习率和 batchsize 是影响模型性能极其重要的两个参数，我们应该非常谨慎地对待。

对于学习率算法，可以选择 Adam 等自适应学习率策略先训练模型看看收敛结果，再考虑使用 SGD 等算法进一步提升性能。对于 Batchsize，大部分人并不会使用几千上万的 batchsize，因此也不用担心模型性能的下降，用大一点(比如 128)的 batchsize 吧，这样需要的迭代次数更少，结果也更加稳定。

【AI 不惑境】残差网络的前世今生与原理

本文是专栏**《AI 不惑境》**的第五篇文章，讲述残差网络的来龙去脉和背后的原理。进入到不惑境界，就是向高手迈进的开始了，在这个境界需要自己独立思考。如果说学习是一个从模仿，到追随，到创造的过程，那么到这个阶段，应该跃过了模仿和追随的阶段，进入了创造的阶段。从这个境界开始，讲述的问题可能不再有答案，更多的是激发大家一起来思考。

作者|言有三

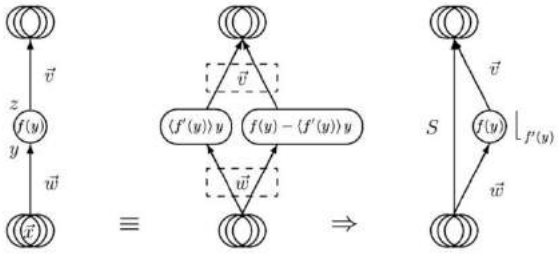
在深度学习模型发展史中，残差网络因其简单而有效的结构与异常有效的结果而占据了非常重要的位置，今天就来仔细说说它的来龙去脉。

1 残差网络之前的历史

残差连接的思想起源于中心化，在神经网络系统中，对输入数据等进行中心化转换，即将数据减去均值，被广泛验证有利于加快系统的学习速度。

$$\forall \mathrm{i}\ \widetilde{x_i} = x_i - \overline{x_i}$$

Schraudolph[1]将这样的思想拓展到了梯度的反向传播中，不仅是输入和隐藏层单元的激活值要中心化，梯度误差以及权重的更新也可以中心化，这便是通过将输入输出进行连接的 shortcut connection，**也称为跳层连接**技术。



在 1998 年的时候，它们提出了将网络分解为 biased 和 centered 两个子网络的思想，通过并行训练两个子网络，分别学习线性和非线性变换部分，不仅降低了各个网络的学习难度，也大大提升了梯度下降算法的训练速度。

Raiko 等人则在论文[2]中更加细致地研究了 shortcut connections 对模型能力的影响，在网络包含 2 到 5 个隐藏层，使用与不使用正则化等各种环境配置下，MNIST 和 CIFAR 图像分类任务和 MNIST 图像重构任务的结果都表明，这样的技术提高了随机梯度下降算法的学习能力，并且提高了模型的泛化能力。

Srivastava 等人在 2015 年的文章[3]中提出了 highway network，对深层神经网络使用了跳层连接，明确提出了残差结构，借鉴了来自于 LSTM 的控制门的思想。

$$y = H(x, W_H) \cdot T(x, W_T) + x \cdot \left(1 - T(x, W_T)\right)$$

当 T(x,Wt)=0 时，y=x，T(x,Wt)=1 时，y=H(x,Wh)T(x,Wt)。在该文章中，研究者没有使用特殊的初始化方法等技巧，也能够训练上千层的网络。

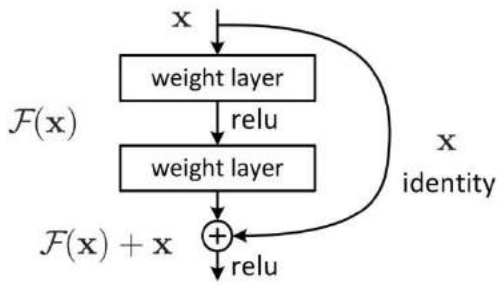
从以上的发展可以看出来，跳层连接由来已久。

2 残差网络

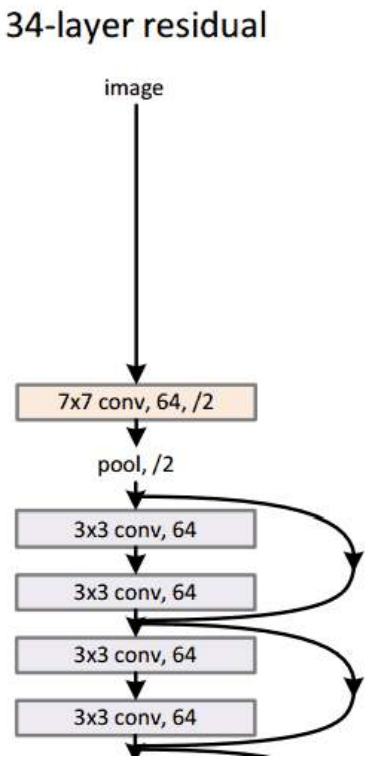
何凯明等人在 2015 年的论文[4]中正式提出了 ResNet，简化了 highway network 中的形式，表达式如下：

$$y = H(x, W_H) + x$$

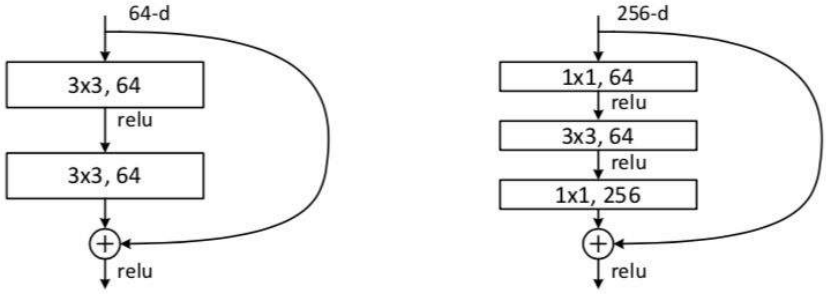
相比于之前的卷积和池化相互堆叠的网络，其基本的结构单元如下：



当我们直接将一个输入添加到输出的时候，输出 y 可以明确的拆分为 H(x,Wh)和 x 的线性叠加，从而让梯度多了一条恒等映射通道，这被认为对于深层网络的训练是非常重要的，一个典型的 resnet 网络结构如下：



resnet 在当年的 ImageNet 的多项竞赛中取得冠军，风头一时无两，随后被广泛深扒。关于 ResNet 的详细解读，大家可以关注我们的往期文章。



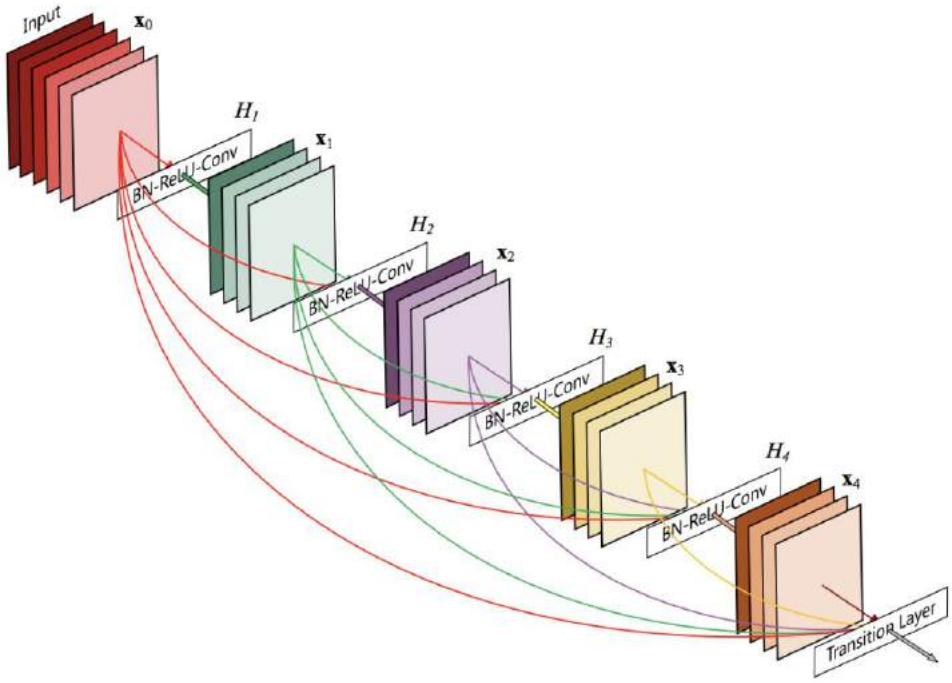
- 【模型解读】resnet 中的残差连接，你确定真的看懂了？

3 残差网络结构的发展

对于残差网络的研究，大部分集中在两个方向，**第一个是结构方面的研究，另一个是残差网络原理的研究，首先说几个具有代表性的结构，不会将所有结构都包含进来，如果感兴趣大家可以关注知识星球有三 AI。**

3.1、更密集的跳层连接 DenseNet

如果将 ResNet 的跳层结构发挥到极致，即每两层都相互连接，那么就成为了 DenseNet，关于 DenseNet 的详细解读，可以查看我们的往期文章。

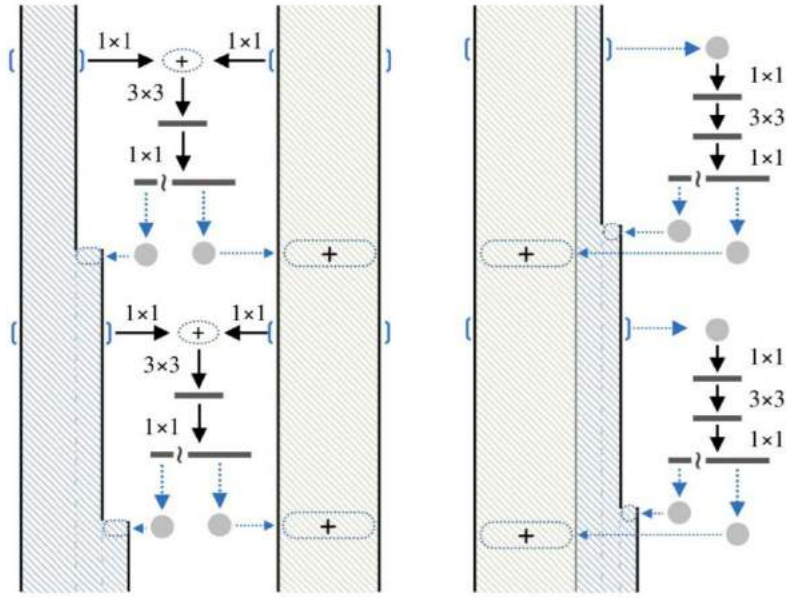


- 【模型解读】“全连接”的卷积网络，有什么好？

DenseNet 是一个非常高效率的网络结构，以更少的通道数更低的计算代价，获得比 ResNet 更强大的性能。

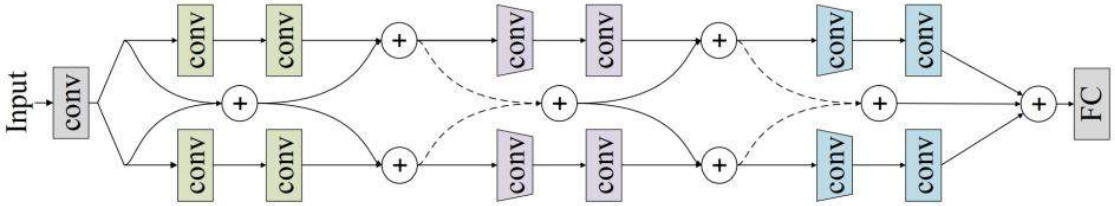
3.2、更多并行的跳层连接

如果将 ResNet 和 DenseNet 分为作为两个通道并行处理，之后再将信息融合，就可以得到 Dual Path Network，网络结构如下：

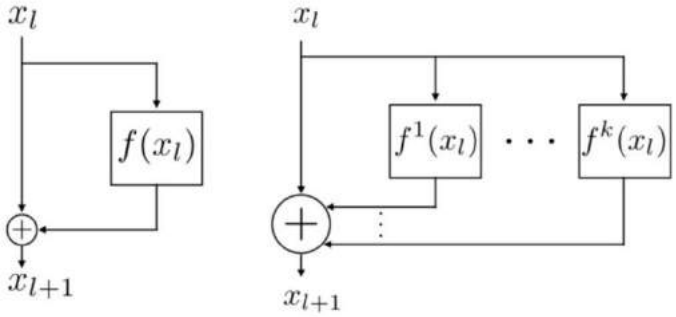


(d) Dual Path Architecture　　(e) DPN

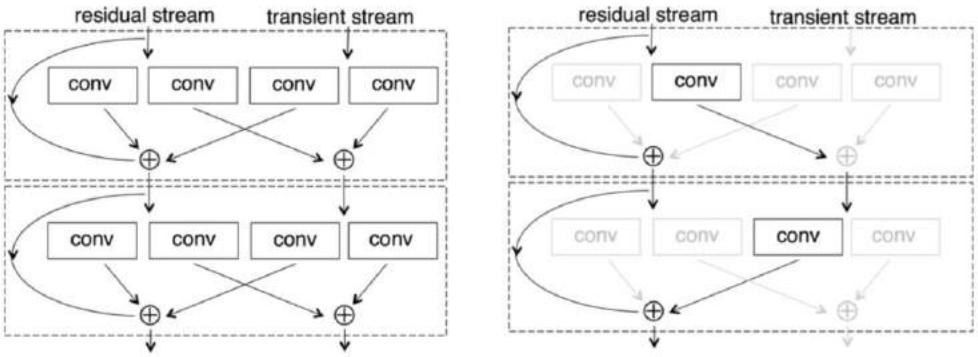
其背后的思想是 resnet 对于重用信息更有效，而 densenet 对于学习新的信息更有效，这个结构在最后一届 ImageNet 竞赛中也取得了很好的成绩，分类比赛的亚军，定位比赛的冠军。类似的结构变种还有如下的结构，不再一一赘述。



当然，还有比上面的 Dual Path Network 更加简单的并行结构，即直接使用多个完全独立且相同的分支并行处理，然后合并。



也有更加复杂的变种，即所谓的 resnet in resnet 结构，如下图。



4 残差网络结构为什么这么好用？

关于残差网络为什么有效，研究众多，这里我们就集中讲述几个主流的思路。

4.1、简化了学习过程，增强了梯度传播

相比于学习原始的信号，残差网络学习的是信号的差值，这在许多的研究中被验证是更加有效的，它简化了学习的过程。

根据我们前面的内容可知，在一定程度上，网络越深表达能力越强，性能越好。

然而随着网络深度的增加，带来了许多优化相关的问题，比如梯度消散，梯度爆炸。

在残差结构被广泛使用之前，研究人员通过研究更好的优化方法，更好的初始化策略，添加 Batch Normalization，提出 Relu 等激活函数的方法来对深层网络梯度传播面临的问题进行缓解，但是仍然不能解决根本问题。

假如我们有这样一个网络：

$$f' = f(x, \omega_f)$$
$$g' = g(f')$$
$$y' = k(g')$$
$$loss = criterion(y, y')$$

其中 f 为卷积操作，g 为非线性变换函数，k 为分类器，依靠误差的链式反向传播法则，损失 loss 对 f 的导数为：

$$\frac{d(f')}{d(\omega_f)} \times \frac{d(g')}{d(f')} \times \frac{d(y')}{d(g')} \times \frac{d(\text{loss})}{d(y')}$$

如果其中某一个导数很小，多次连乘后梯度可能越来越小，这就是常说的梯度消散，对于深层网络，从靠近输出的深层传到靠近输入的浅层时梯度值非常小，使得浅层无法有效地更新。

如果使用了残差结构，因为导数包含了恒等项，仍然能够有效的反向传播。 举一个非常直观的例子方便理解，假如有一个网络，输入 x=1，非残差网络为 G，残差网络为 H，其中 H(x)=F(x)+x，假如有这样的输入关系：

在 t 时刻：

非残差网络$G_t(1) = 1.1$

残差网络$H_t(1) = 1.1,\ H_t(1) = F_t(1) + 1, F_t(1) = 0.1$

在 t+1 时刻：

非残差网络$G_{t+1}(1) = 1.2$

残差网络$H_{t+1}(1) = 1.2, H_{t+1}(1) = F_{t+1}(1) + 1, F_{t+1}(1) = 0.2$

非残差网络 G 的梯度$=(G_{t+1}(1)-G_t(1))/G_t(1)) = (1.2 - 1.1)/1.1$

而残差网络 F 的梯度$=(F_{t+1}(1)-F_t(1))/F_t(1)) = (0.2 - 0.1)/0.1$

因为两者各自是对 G 的参数和 F 的参数进行更新，可以看出变化对 F 的影响远远大于 G，说明引入残差后的映射对输出的变化更敏感，这样是有利于网络进行传播的。

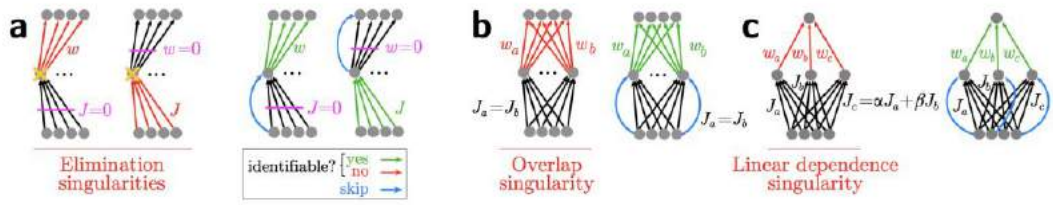
4.2、打破了网络的不对称性[5]

虽然残差网络可以通过跳层连接，增强了梯度的流动，从而使得上千层网络的训练成为可能，**不过相关的研究表面残差网络的有效性，更加体现在减轻了神经网络的退化。**

如果在网络中每个层只有少量的隐藏单元对不同的输入改变它们的激活值，而大部分隐藏单元对不同的输入都是相同的反应，此时整个权重矩阵的秩不高。并且随着网络层数的增加，连乘后使得整个秩变的更低，这就是我们常说的网络退化问题。

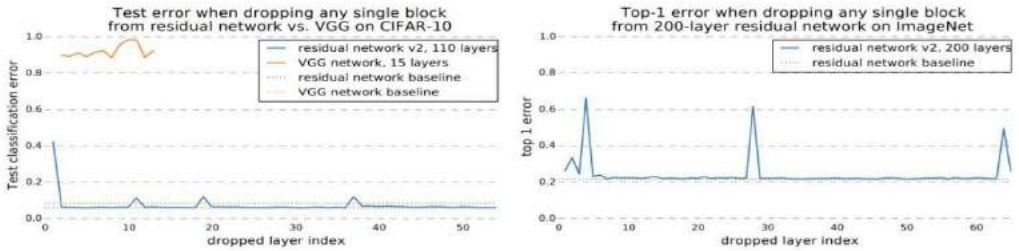
虽然权重矩阵是一个很高维的矩阵，但是大部分维度却没有信息，使得网络的表达能力没有看起来那么强大。这样的情况一定程度上来自于网络的对称性，而残差连接打破了网络的对称性。

下面展示了三种跳层连接恢复网络表达能力的案例，分别是**消除输入和权重零奇点，打破对称性，线性依赖性**。

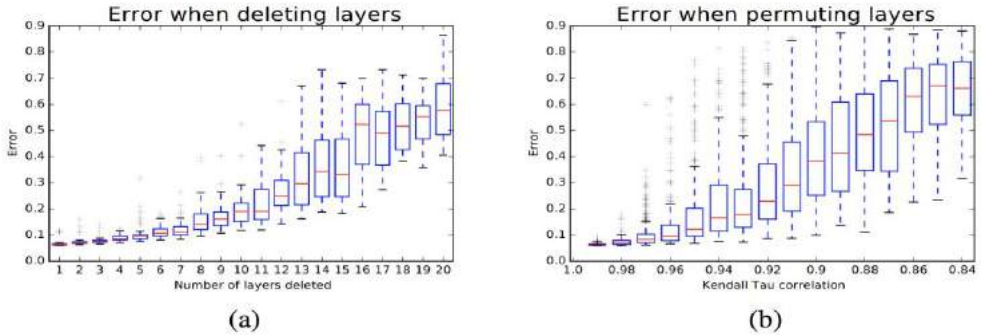


4.3、增强了网络的泛化能力[6]

有一些研究表明，深层的残差网络可以看做是不同深度的浅层神经网络的 ensemble，训练完一个深层网络后，在测试的时候随机去除某个网络层，并不会使得网络的性能有很大的退化,而对于 VGG 网络来说，删减任何一层都会造成模型的性能奔溃，如下图。



甚至去除和打乱一些网络层，性能的下降也是一个很平滑的过程。



(a) (b)

以上都证明了残差结构其实是多个更浅的网络的集成，所以它的有效深度看起来表面的那么深，因此优化自然也没有那么难了。

关于残差，还有需要的研究，大家可以持续关注我们公众号和知乎，以及星球。

参考文献

[1] Schraudolph N. Accelerated gradient descent by factor-centering decomposition[J]. Technical report/IDSIA, 1998, 98.

[2] Raiko T, Valpola H, LeCun Y. Deep learning made easier by linear transformations in perceptrons[C]//Artificial intelligence and statistics. 2012: 924-932.

[3] Srivastava R K, Greff K, Schmidhuber J. Training very deep networks[C]//Advances in neural information processing systems. 2015: 2377-2385.

[4] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[5] Orhan A E, Pitkow X. Skip Connections Eliminate Singularities[J]. international conference on learning representations, 2018.
[6] Veit A, Wilber M J, Belongie S. Residual networks behave like ensembles of relatively shallow networks[C]//Advances in neural information processing systems. 2016: 550-558.

5 总结

通过跳层连接而来的残差网络虽然结构非常简单，但是却异常有用，在很多的应用领域中都被广泛使用，其原理也在被进一步广泛研究中，大家可以持续关注，

【AI 不惑境】移动端高效网络，卷积拆分和分组的精髓

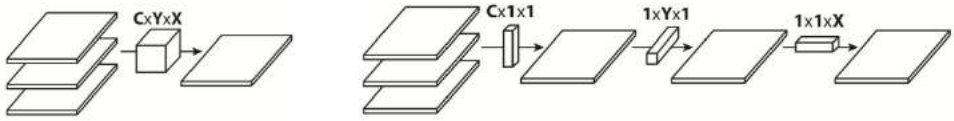
本文是《AI 不惑境》的第六篇文章，讲述卷积拆分和分组卷积的精髓。进入到不惑境界，就是向高手迈进的开始了，在这个境界需要自己独立思考。如果说学习是一个从模仿，到追随，到创造的过程，那么到这个阶段，应该跃过了模仿和追随的阶段，进入了创造的阶段。从这个境界开始，讲述的问题可能不再有答案，更多的是激发大家一起来思考。

作者|言有三

在移动端高效的模型设计中，卷积拆分和分组几乎是不可缺少的思想，那么它们究竟是如何高效，本身又有哪些发展呢。

1 什么是卷积拆分

一个多通道的普通 2D 卷积包含了三个维度，分别是通道，长，宽，如下图(a)。



(a) 3D convolution　　(b) 1D convolutions over different directions

然后将这个卷积的步骤分解为 3 个独立的方向[1]，即通道方向，X 方向和 Y 方向，如上图(b)，则具有更低的计算量和参数量。

假如 X 是卷积核宽度，Y 是卷积核高度，C 是输入通道数，如果是正常的卷积，那么输出一个通道，需要的参数量是 XYC，经过上图的分解后，参数量变为 X+Y+C，一般来说 C>>X 和 Y，所以分解后的参数对比之前的参数约为 1/(XY)。

对于 3×3 的卷积，相当于参数量降低一个数量级，计算量也是相当，可见这是很高效的操作。

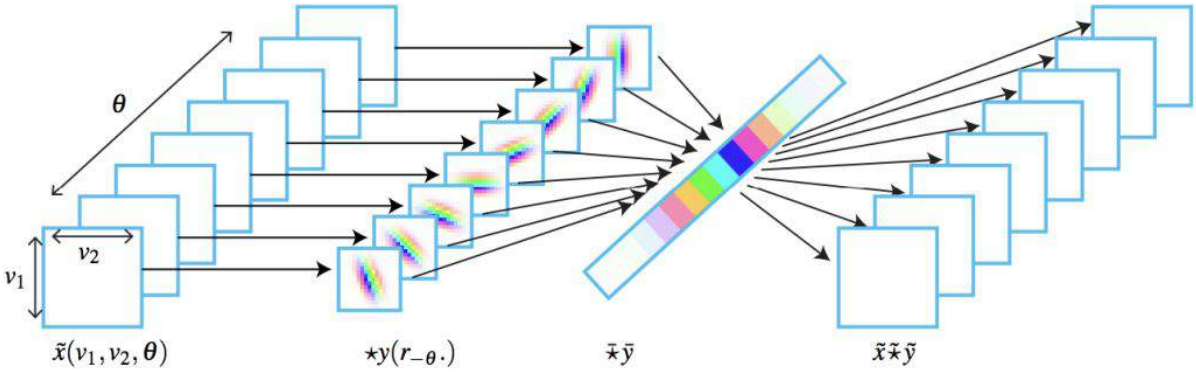
当然，还可以只分解其中的某些维度，比如在 Inception V3 的网络结构中，就将 7×7 的卷积拆分为 1×7 和 7×1 两个方向。从另一个角度来看，这还提升了网络的深度。

2 什么是通道分组

2.1 分组卷积的来源

标准的卷积是使用多个卷积核在输入的所有通道上分别卷积提取特征，而分组卷积，就是将通道进行分组，组与组之间相关不影响，各自得到输出。

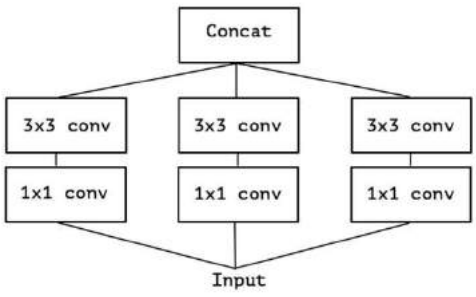
通道分组的思想来自于 Laurent Sifre 在 Google 实习的时候提出的 separable convolution，相关的内部报告可以参考 YouTube 视频 https://www.youtube.com/watch?v=VhLe-u0M1a8，具体的实现在它的博士论文[2]中，如下示意图。



对于平移，旋转等刚体运动来说，它们可以被拆分成不同的维度，因此使用上面的 separable convolution，实现起来也很简单，就是先进行通道的分组，这在 AlexNet 网络中还被当作一个训练技巧。

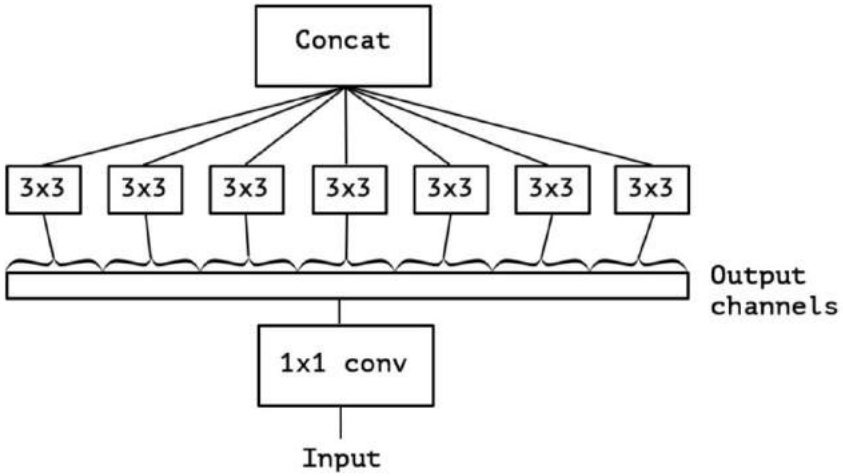
2.2 从 Xception 到 MobileNet

随着 Google 的 Inception 网络提出，这一个相对于 VGG 更加高效的网络也开始进化。到了 Inception V2 的时候，已经用上了上面的思想。



上面就是一个与 Inception Module 类似的模块，只是每一个通道完全一样，这就可以等价于通道分组了。

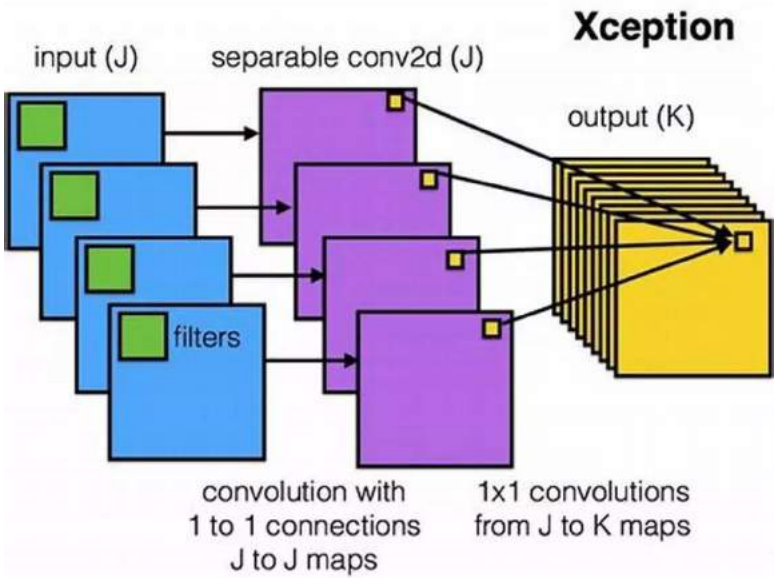
假如分组的个数与输入通道数相等，Inception 便成为了极致的 inception(extreme inception，简称 Xception[3])。



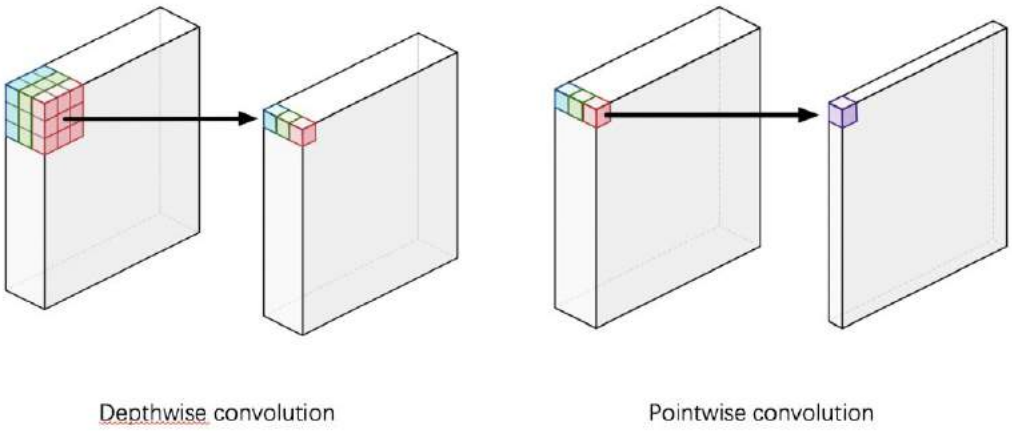
首先经过 1×1 卷积，然后通道分组进行卷积，这样的一个结构随 Tensorflow 的流行而流行，名为 Depthwise Separable Convolution。

随后 Google 的研究人员提出了 MobileNets[4]结构，使用了 Depthwise Separable Convolution 模块进行堆叠，与 Xception 中的不同是 1×1 卷积放置在分组卷积之后。因为有许多这样的模块进行堆叠，所以两者其实是等价的。

画成二维图，示意图如下：



画成三维图，示意图如下：



使用 Netscope 可视化 MobileNet 的网络如下，当我们看到这个 28 层的网络，又经历了残差网络的洗礼后，顿时有种返璞归真的感觉。

2.3 分组卷积性能如何

令输入 blob 大小为 M×Dk×Dk，输出为 N×Dj×Dj，则标准卷积计算量为 M×Dk×Dk×N×Dj×Dj，而转换为 Depthwise 卷积加 Pointwise 卷积，Depthwise 卷积计算量为 M×Dk×Dk×Dj×Dj，Pointwise 卷积计算量为 M×N×Dj×Dj，计算量对比为：(M×Dk×Dk×Dj×Dj+M×N×Dj×Dj)/M×Dk×Dk×N×Dj×Dj= 1/N+1/(Dk×Dk)，由于网络中大量地使用 3×3 的卷积核，当 N 比较大时，上面卷积计算量约为普通卷积的 1/9，从而降低了一个数量级的计算量。

性能上也没有让我们失望，在只有 VGG16 不到 1/32 的参数量和 1/27 的计算量的同时还能取得与之相当的性能。

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

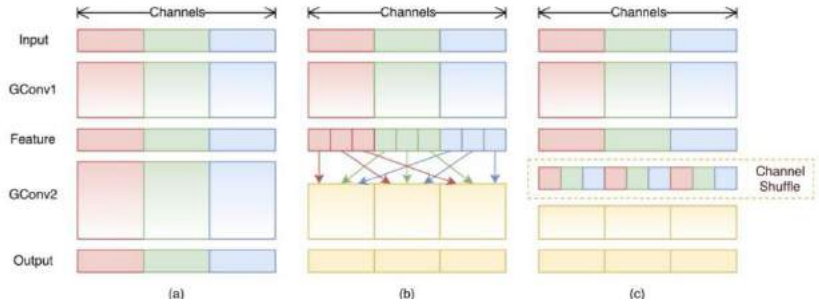
关于更多细节的解读和实验对比，此处就不再做介绍了，可以阅读以前的一篇文章。

- 【模型解读】说说移动端基准模型 MobileNets

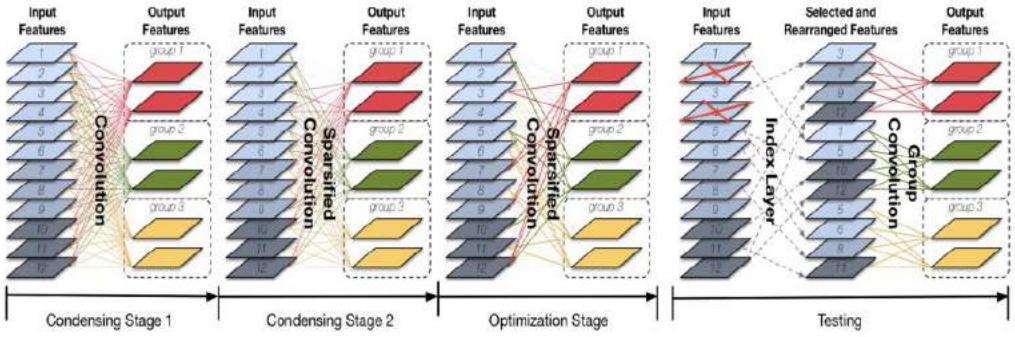
2.4 分组卷积性能的进一步提升

对于 MobileNet 这样的网络结构，还可以从两个方向进行提升，第一个是增加分组的信息交流，第二个是更加智能的分组。

简单的分组使得不同通道之间没有交流，可能会导致信息的丢失，Shufflenet[5]重新增加了通道的信息交换。具体来说，对于上一层输出的通道，先做一个 Shuffle 操作，再分成几个组进入到下一层，示意图如下：



另一方面，MobileNet 的分组是固定，ShuffleNet 中的通道的打乱也是一个确定的映射，那是不是可以基于数据来学习到更加合适的分组呢？Condensenets[6]给出了确定的回答。



更多的解读，我们已经放在了知识星球中，感兴趣的可以关注。

3 分组卷积结构的发展

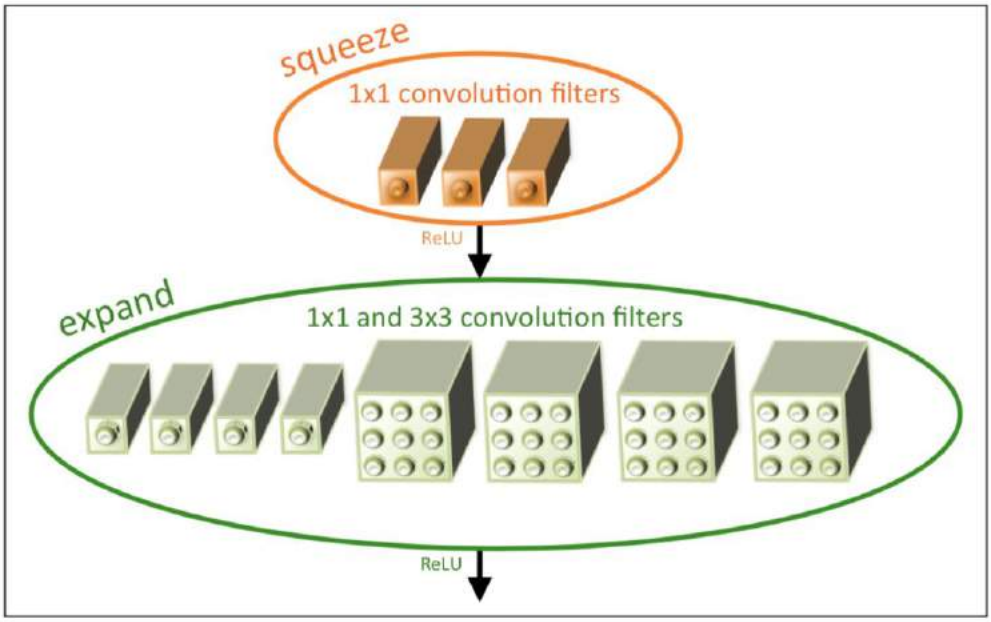
ResNet 虽然不是残差连接的发明者，但使得这一思想为众人痴狂。MobileNet 也不是分组卷积的发明者，但同样是它使分组的思想深入人心，原来这样的网络结构不仅不降低准确率，还能大幅度提升计算效率，尤其适合硬件并行。

自此，分组的思想被不断拓展研究，下面我们主要考虑分组的各个通道存在较大差异的研究。

3.1 多分辨率卷积核通道分组网络

这一类网络以 SqueezeNet[7]为代表，它以卷积层 conv1 开始，接着是 8 个 Fire modules，最后以卷积层 conv10 结束。

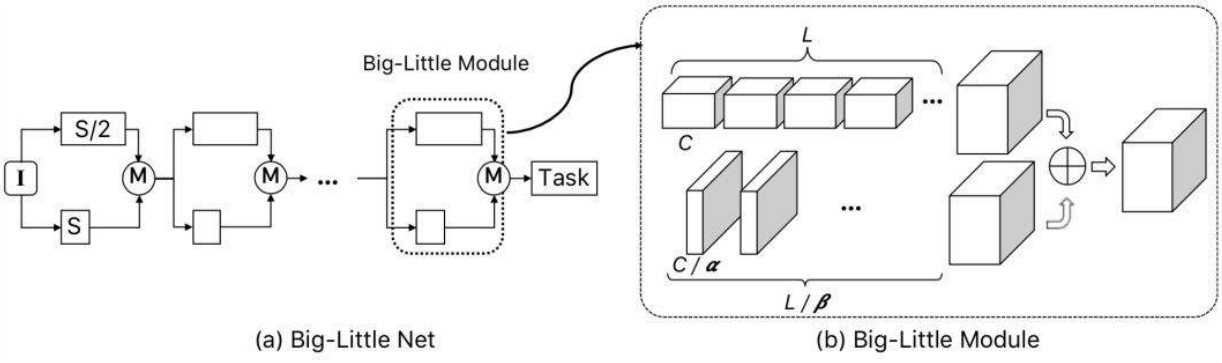
一个 fire module 的子结构下图，包含一个 squeeze 模块加上一个 expand 模块。Squeeze 模块使用 1×1 卷积进行通道降维，expand 模块使用 1×1 卷积和 3×3 卷积用于通道升维。



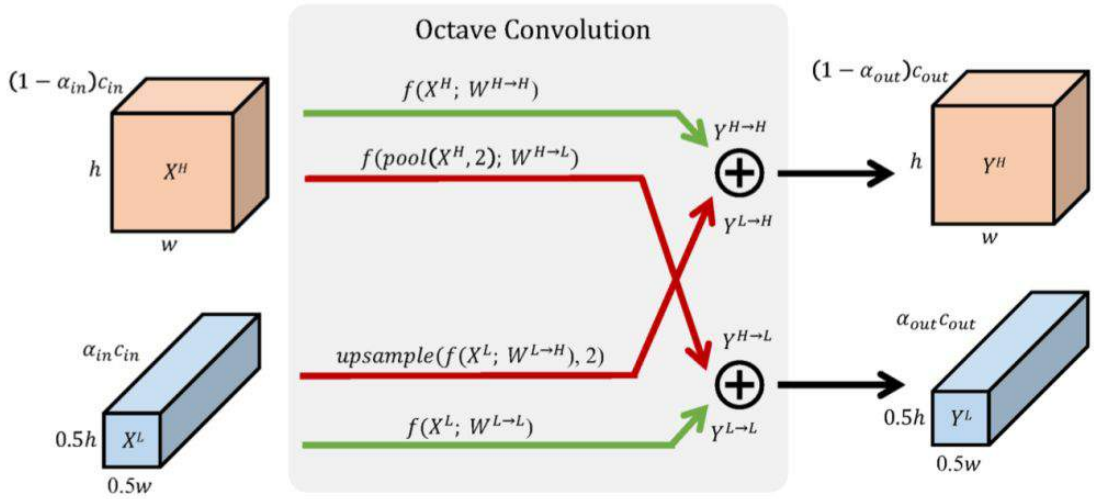
Squeezenet 的压缩比率是惊人的，只有 AlexNet 1/50 的参数量，能达到相当的性能。

3.2 多尺度通道分组网络

这一类结构采用不同的尺度对信息进行处理，对于分辨率大的分支，使用更少的卷积通道，对于分辨率小的分支，使用更多的卷积通道，以 Big-Little Net[8]为代表，K 个分支，尺度分别为 1/2^(K-1)，如下图结构。



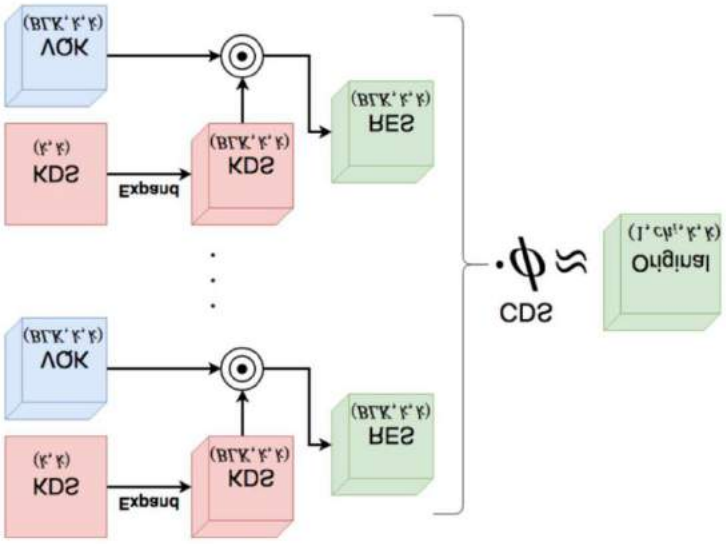
当然，如果两个通道在中间的计算过程中还存在信息的交流，则可以获得更高的性能，比如 Octave Convolution[9]。



卷积核通过因子被分为了高分辨率和低分辨率两部分，低分辨率具有较多的通道，被称为低频分量。高分辨率具有较少的通道，被称为高频分量，两者各自学习，并且进行信息的融合。高分辨率通道通过池化与低分辨率通道融合，低分辨率通过上采样与高分辨率通道融合。最终在 22.2GFLOPS 的计算量下，ImageNet Top-1 的精度达到了 82.9%。

3.3 多精度通道分组网络

除了还分辨率和卷积核上做文章，还可以在计算精度上做文章，这一类结构以 DSConv[10]为代表，它将卷积核分为两部分，一部分是整数分量 VQK，一部分是分数分量 KDS，如下图：



VQK(Variable Quantizes Kernel)只有整数值，不可训练，它的权重值从预训练模型中计算而来。KDS(kernel distribution shifter)是浮点数，包含一个 kernel 级别的偏移量，一个 channel 级别的偏移量。

这一个模型在 ResNet50 和 ResNet34，AlexNet，MobileNet 等基准模型上取得了 14x 参数量的压缩，10x 速度的提升。

除了上面这些思路外，还有很多可以做的空间，大家可以去多实验写论文填坑，有三就帮到这里了。

参考文献

[1] Jin J, Dundar A, Culurciello E. Flattened convolutional neural networks for feedforward acceleration[J]. arXiv preprint arXiv:1412.5474, 2014.

[2] Sifre L , Mallat, Stéphane. Rigid-Motion Scattering for Texture Classification[J]. Computer Science, 2014.

[3] Chollet F. Xception: Deep Learning with Depthwise Separable Convolutions[J]. computer vision and pattern recognition, 2017: 1800-1807.

[4] Howard A G, Zhu M, Chen B, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications[J]. arXiv preprint arXiv:1704.04861, 2017.

[5] Zhang X, Zhou X, Lin M, et al. Shufflenet: An extremely efficient convolutional neural network for mobile devices[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018: 6848-6856.
[6] Huang G, Liu S, Van der Maaten L, et al. Condensenet: An efficient densenet using learned group convolutions[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018: 2752-2761.
[7] Iandola F N, Han S, Moskewicz M W, et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size[J]. arXiv preprint arXiv:1602.07360, 2016.
[8] Chen C F, Fan Q, Mallinar N, et al. Big-little net: An efficient multi-scale feature representation for visual and speech recognition[J]. arXiv preprint arXiv:1807.03848, 2018.
[9] Chen Y, Fang H, Xu B, et al. Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution[J]. arXiv preprint arXiv:1904.05049, 2019.
[10] Gennari M, Fawcett R, Prisacariu V A. DSConv: Efficient Convolution Operator[J]. arXiv preprint arXiv:1901.01928, 2019.

4、总结

分组卷积之所有有效，一个是因为网络中的空间和通道的冗余计算使得其性能可以保持，而简单的分组并行计算又非常适合于 GPU 等处理器，因此在移动端高效率模型中广泛使用，是必须掌握的思想。

【AI 不惑境】深度学习中的多尺度模型设计

本文是专栏**《AI 不惑境》**的第七篇文章，讲述计算机视觉中的多尺度问题。进入到不惑境界，就是向高手迈进的开始了，在这个境界需要自己独立思考。如果说学习是一个从模仿，到追随，到创造的过程，那么到这个阶段，应该跃过了模仿和追随的阶段，进入了创造的阶段。从这个境界开始，讲述的问题可能不再有答案，更多的是激发大家一起来思考。

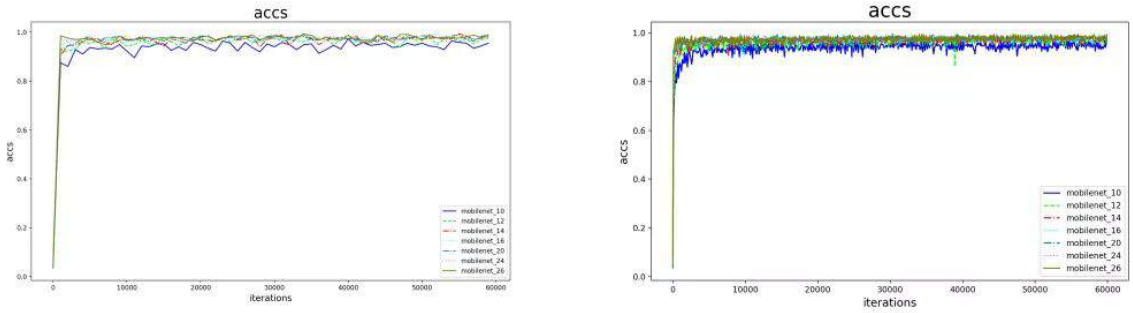
作者|言有三

在计算机视觉中，尺度始终是一个大问题，小物体与超大尺度物体往往都会严重影响性能。

1 什么是多尺度

1.1 什么是多尺度

所谓多尺度，实际上就是对**信号的不同粒度的采样**，通常在不同的尺度下我们可以观察到不同的特征，从而完成不同的任务。



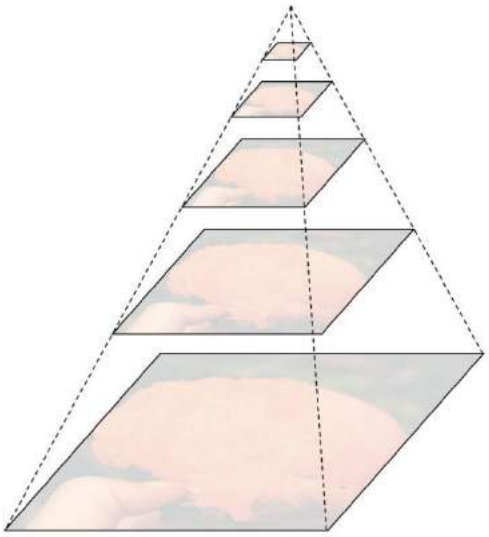
如上两个图是同样的一维信号在不同采样频率下的结果，这是一条精度曲线。通常来说粒度更小/更密集的采样可以看到更多的细节，粒度更大/更稀疏的采样可以看到整体的趋势，不过此处由于使用了不同的颜色，曲线本身也存在较大的波动，所以粒度更小的右图反而能更直观的看到各个曲线的整体性能比较结果

如上展示了 3 个尺度的图像，如果要完成的任务只是判断图中是否有前景，那么 12×8 的图像尺度就足够了。如果要完成的任务是识别图中的水果种类，那么 64×48 的尺度也能勉强完成。如果要完成的任务是后期合成该图像的景深，则需要更高分辨率的图像，比如 640×480。

1.2 图像金字塔

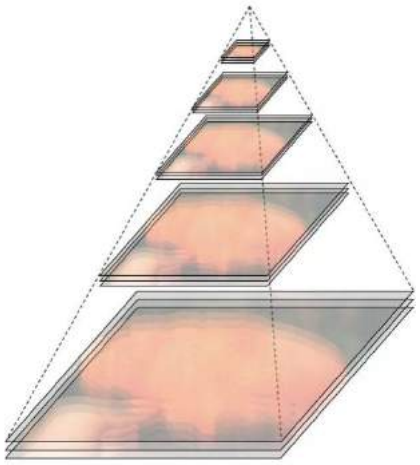
很多时候多尺度的信号实际上已经包含了不同的特征，为了获取更加强大的特征表达，在传统图像处理算法中，有一个很重要的概念，即图像金字塔和高斯金字塔。

图像金字塔，即一组不同分辨率的图像，如下图，



采样的方式可以是不重叠或者重叠的，如果是不重叠的，采样尺度因子为 2，那就是每增加一层，行列分辨率为原来的 1/2。

当然，为了满足采样定理，每一个采样层还需要配合平滑滤波器，因此更常用的就是高斯金字塔，每一层内用了不同的平滑参数，在经典的图像算子 SIFT 中被使用。



不过这不是本文要聚焦的内容，请大家去自行了解**尺度空间理论**，接下来聚焦深度学习中的多尺度模型设计。

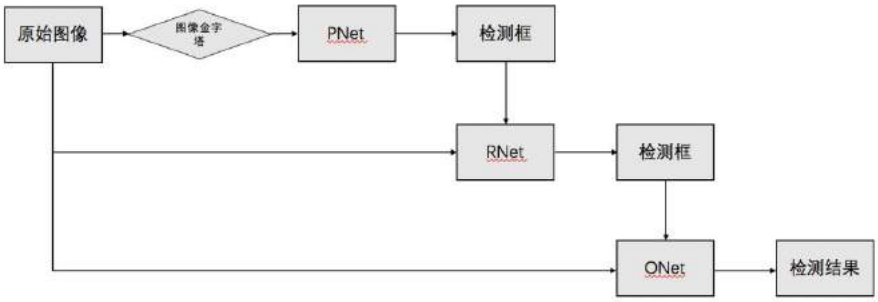
2 计算机视觉中的多尺度模型架构

卷积神经网络通过逐层抽象的方式来提取目标的特征，其中一个重要的概念就是感受野。如果感受野太小，则只能观察到局部的特征，如果感受野太大，则获取了过多的无效信息，因此研究人员一直都在设计各种各样的多尺度模型架构，主要是图像金字塔和特征金字塔两种方案，但是具体的网络结构可以分为以下几种：(1) 多尺度输入。(2) 多尺度特征融合。(3) 多尺度特征预测融合。(4) 以上方法的组合。

2.1 多尺度输入网络

顾名思义，就是使用**多个尺度的图像输入(图像金字塔)**，然后将其结果进行融合，传统的人脸检测算法 **V-J 框架**就采用了这样的思路。

深度学习中模型以 MTCNN[1]人脸检测算法为代表，其流程如下，在第一步检测 PNet 中就使用了多个分辨率的输入，各个分辨率的预测结果(检测框)一起作为RNet 的输入。



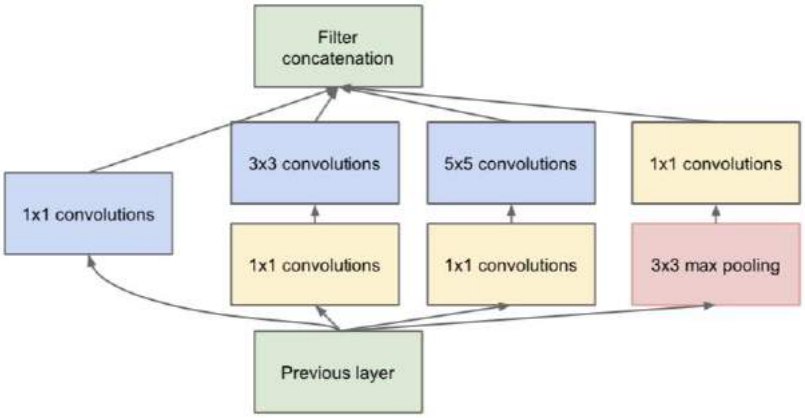
值得一提的是，**多尺度模型集成的方案**在提高分类任务模型性能方面是不可或缺的，许多的模型仅仅采用多个尺度的预测结果进行**平均值融合**，就能在 ImageNet 等任务中提升 2%以上的性能。

2.2 多尺度特征融合网络

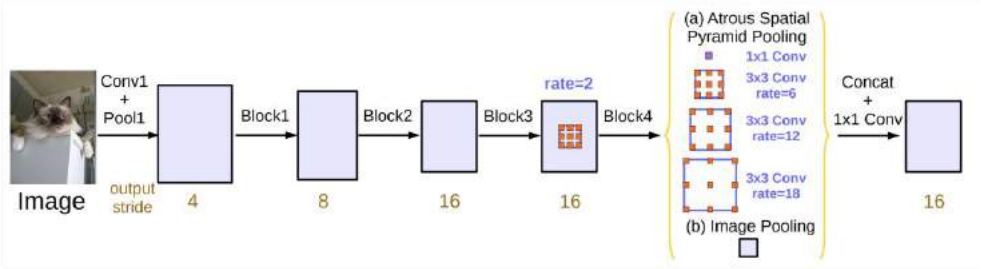
多尺度特征融合网络常见的有两种，第一种是**并行多分支网络**，第二种是**串行的跳层连接结构，都是在不同的感受野下进行特征提取**。

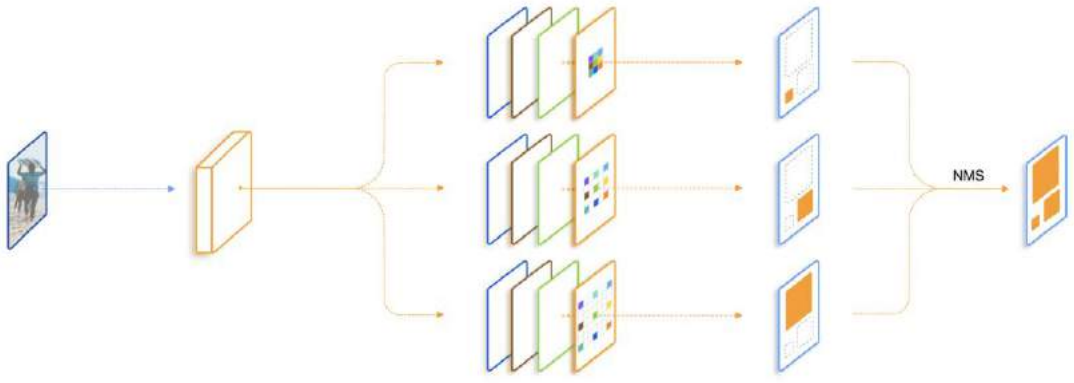
(1) 并行多分支结构

比如 Inception 网络中的 Inception 基本模块，包括有四个并行的分支结构，分别是 1×1 卷积，3×3 卷积，5×5 卷积，3×3 最大池化，最后对四个通道进行组合。

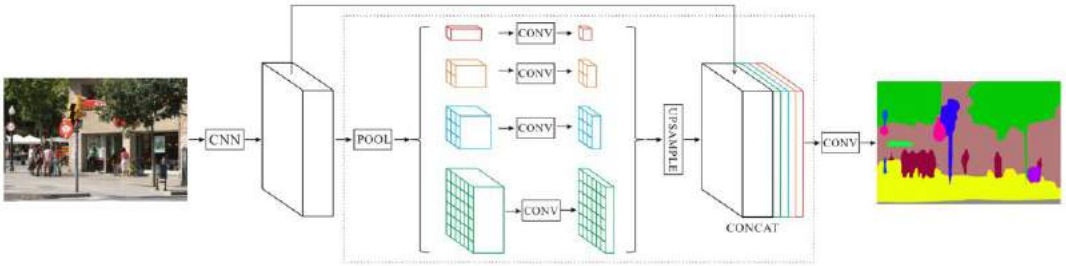


除了更高卷积核大小，还可以使用带孔卷积来控制感受野。在图像分割网络 Deeplab V3[2]和目标检测网络 trident networks[3]中都使用了这样的策略，网络结构如下图：

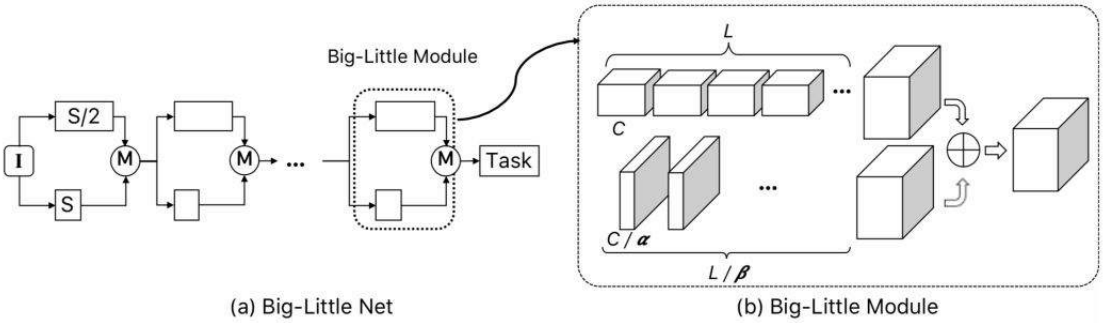




还有一种比不同大小的卷积核和带孔卷积计算代价更低的控制感受野的方法，即直接使用不同大小的池化操作，被 PSPNet[4]采用。



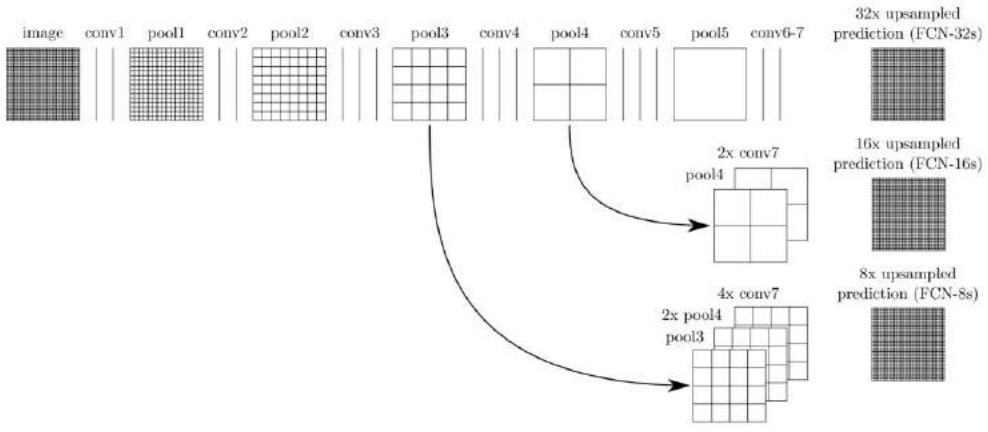
值得注意的是，这样的多分支结构对于模型压缩也是有益处的，以 Big-little Net[5]为代表，它采用不同的尺度对信息进行处理。



对于分辨率大的分支，使用更少的卷积通道，对于分辨率小的分支，使用更多的卷积通道，这样的方案能够更加充分地使用通道信息。

(2) 串行多分支结构

串行的多尺度特征结构以 FCN[6]，U-Net 为代表，需要通过**跳层连接**来实现特征组合，这样的结构在图像分割/目标检测任务中是非常常见的。

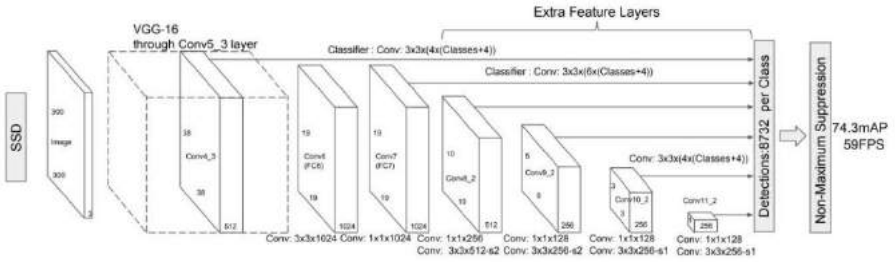


从上面这些模型可以看出，并行的结构能够在同一层级获取不同感受野的特征，经过融合后传递到下一层，可以更加灵活地平衡计算量和模型能力。串行的结构将不同抽象层级的特征进行融合，对于边界敏感的图像分割任务是不可缺少的。

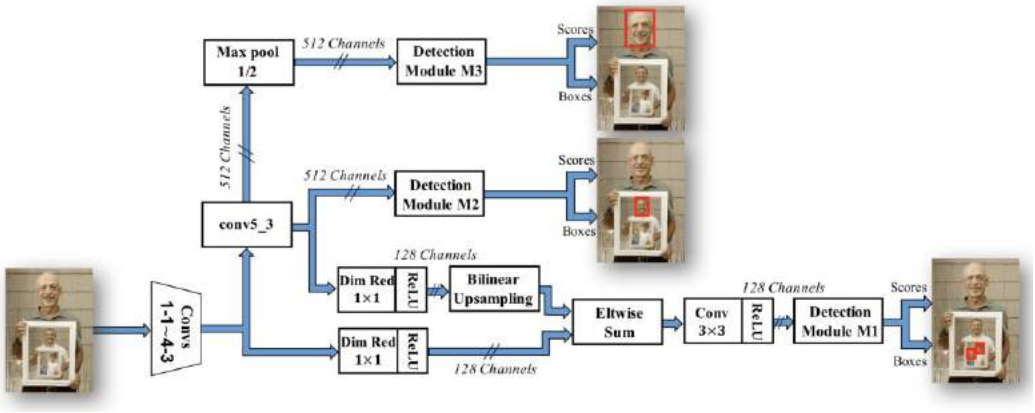
2.3 多尺度特征预测融合

即在**不同的特征尺度进行预测**，最后将结果进行融合，以目标检测中的 SSD[7]为代表。

SSD 在不同 stride 不同大小的特征图上进行预测。低层特征图 stride 较小，尺寸较大，感受野较小，期望能检测到小目标。高层特征图stride较大，尺寸较小，感受野较大，期望能检测到大目标。



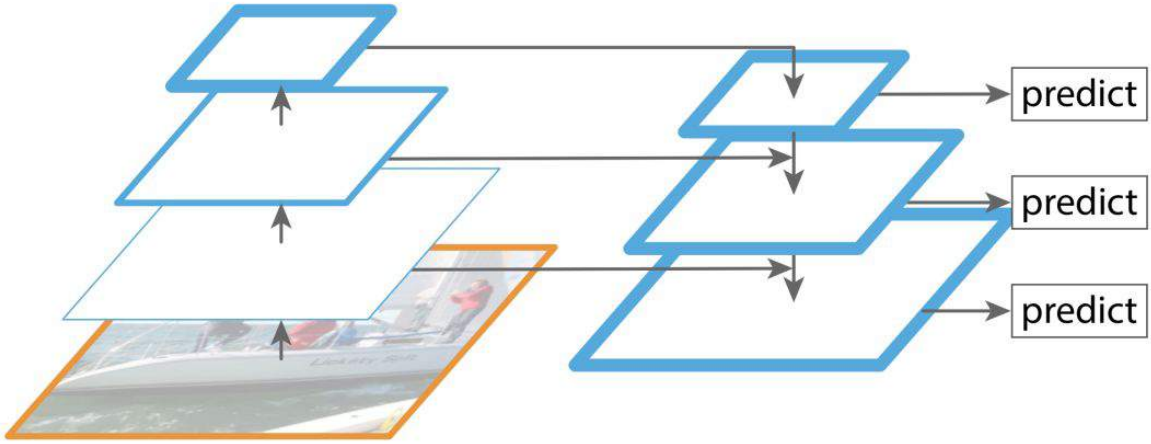
类似的思想还有 SSH[8]，从分辨率较大的特征图开始分为多个分支，然后各个分支单独预测不同尺度大小的目标。



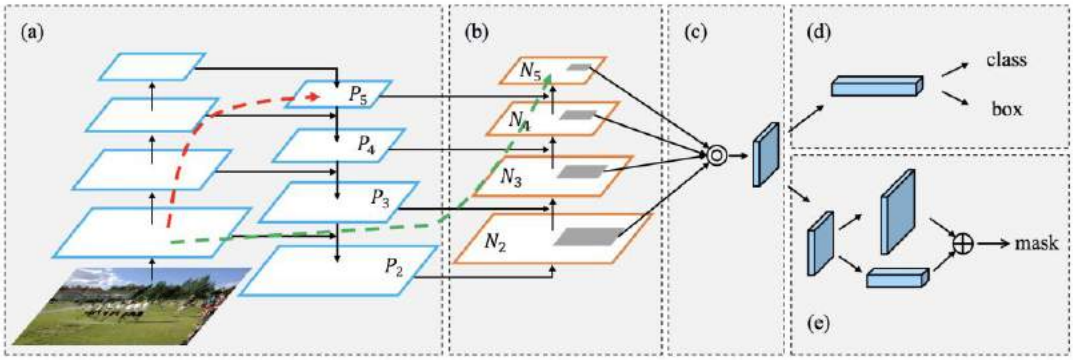
在多个特征通道进行预测的思想与多个输入的方案其实是异曲同工的，但是它的计算效率更高。

2.4 多尺度特征和预测融合

既然可以将不同尺度的特征进行融合，也可以在不同的尺度进行预测，为何不同时将这两种机制一起使用呢？这样的结构以目标检测中的 FPN[9]为代表。



即将高层的特征添加到相邻的低层组合成新的特征，每一层单独进行预测。当然，也可以反过来将低层的特征也添加到高层，比如 PAN[10]。



当然，对于不同尺度的特征图的融合，还可以基于学习的融合方案。

3 后话

上面说了这么多的方法，相信动手能力强的同学一定可以基于这些方法进行排列组合和拓展。另外，对于某些领域中的专用技巧，比如**目标检测中的不同尺度的 Anchor 设计，不同尺度的训练技巧**，在这里没有讲述。

以上就是多尺度的常用设计方法，从图像分辨率的控制，到卷积核，池化的大小和不同的方案，到不同特征的融合，现在有很多相关的研究，本文不一一详述，作为计算机视觉中的老大难问题，我们会持续关注，相关自动架构搜索的研究也已经出现[11]。

参考文献

[1] Zhang K, Zhang Z, Li Z, et al. Joint face detection and alignment using multitask cascaded convolutional networks[J]. IEEE Signal Processing Letters, 2016, 23(10): 1499-1503.

[2] Chen L C, Papandreou G, Schroff F, et al. Rethinking atrous convolution for semantic image segmentation[J]. arXiv preprint arXiv:1706.05587, 2017.

[3] Li Y, Chen Y, Wang N, et al. Scale-aware trident networks for object detection[J]. arXiv preprint arXiv:1901.01892, 2019.

[4] Zhao H, Shi J, Qi X, et al. Pyramid scene parsing network[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 2881-2890.

[5] Chen C F, Fan Q, Mallinar N, et al. Big-little net: An efficient multi-scale feature representation for visual and speech recognition[J]. arXiv preprint arXiv:1807.03848, 2018.

[6]Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 3431-3440.

[7] Liu W, Anguelov D, Erhan D, et al. Ssd: Single shot multibox detector[C]//European conference on computer vision. Springer, Cham, 2016: 21-37.

[8] Najibi M, Samangouei P, Chellappa R, et al. Ssh: Single stage headless face detector[C]//Proceedings of the IEEE International Conference on Computer Vision. 2017: 4875-4884.

[9] Lin T Y, Dollár P, Girshick R, et al. Feature pyramid networks for object detection[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 2117-2125.

[10] Liu S, Qi L, Qin H, et al. Path aggregation network for instance segmentation[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018: 8759-8768.

[11] Ghiasi G, Lin T Y, Le Q V. Nas-fpn: Learning scalable feature pyramid architecture for object detection[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019: 7036-7045.

4 总结

多尺度不仅对检测和分割不同尺度的目标很重要，对于提高模型的参数使用效率也非常关键，是必须深刻理解和掌握的方法。

【AI 不惑境】计算机视觉中注意力机制原理及其模型发展和应用

本文是专栏**《AI 不惑境》**的第八篇文章，讲述计算机视觉中的注意力(attention)机制。进入到不惑境界，就是向高手迈进的开始了，在这个境界需要自己独立思考。如果说学习是一个从模仿，到追随，到创造的过程，那么到这个阶段，应该跃过了模仿和追随的阶段，进入了创造的阶段。从这个境界开始，讲述的问题可能不再有答案，更多的是激发大家一起来思考。

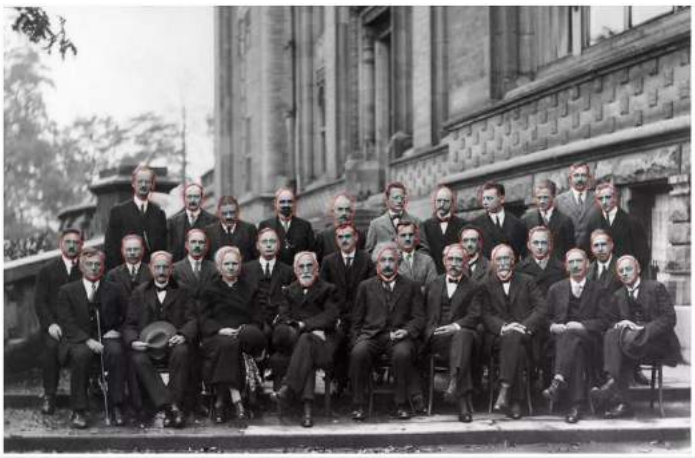
作者|言有三

Attention 机制在近几年来在图像，自然语言处理等领域中都取得了重要的突破，被证明有益于提高模型的性能。Attention 机制本身也是符合人脑和人眼的感知机制，这次我们主要以计算机视觉领域为例，讲述 Attention 机制的原理，应用以及模型的发展。

1 Attention 机制与显著图

1.1 何为 Attention 机制

所谓 Attention 机制，便是聚焦于局部信息的机制，比如图像中的某一个图像区域。随着任务的变化，注意力区域往往会发生变化。



面对上面这样的一张图，如果你只是从整体来看，只看到了很多人头，但是你拉近一个一个仔细看就了不得了，都是天才科学家。

图中除了人脸之外的信息其实都是无用的，也做不了什么任务，**Attention 机制便是要找到这些最有用的信息**，可以想见最简单的场景就是从照片中检测人脸了。

1.2 基于 Attention 的显著目标检测

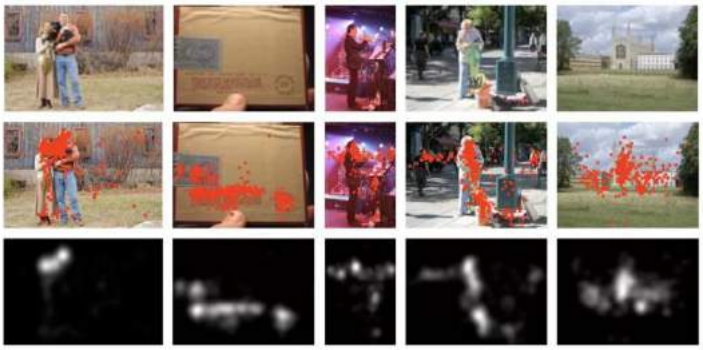
和注意力机制相伴而生的一个任务便是显著目标检测，即 salient object detection。它的输入是一张图，输出是一张概率图，概率越大的地方，代表是图像中重要目标的概率越大，即人眼关注的重点，一个典型的显著图如下：

右图就是左图的显著图，在头部位置概率最大，另外腿部，尾巴也有较大概率，这就是图中真正有用的信息。

显著目标检测需要一个数据集，而这样的数据集的收集便是通过追踪多个实验者的眼球在一定时间内的注意力方向进行平均得到，典型的步骤如下：

(1) 让被测试者观察图。
(2) 用 eye tracker 记录眼睛的注意力位置。
(3) 对所有测试者的注意力位置使用高斯滤波进行综合。
(4) 结果以 0～1 的概率进行记录。

于是就能得到下面这样的图，第二行是眼球追踪结果，第三行就是显著目标概率图。



上面讲述的都是空间上的注意力机制，即关注的是不同空间位置，而在 CNN 结构中，还有不同的特征通道，因此不同特征通道也有类似的原理，下面一起讲述。

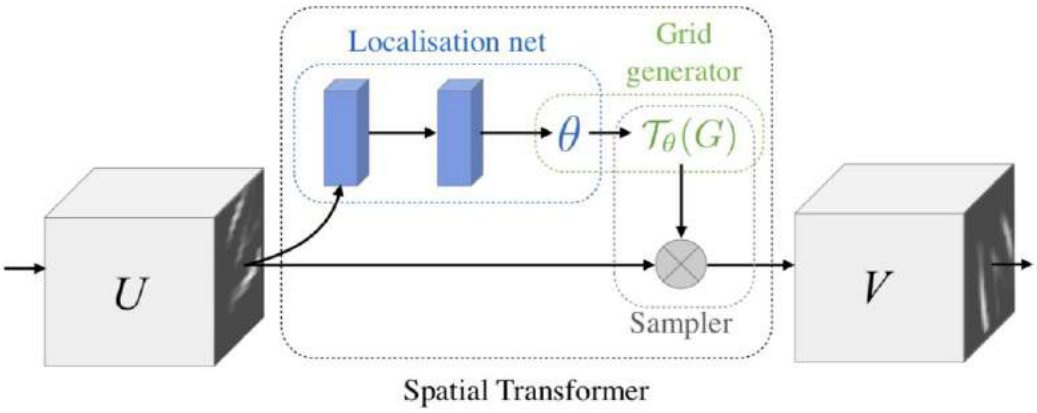
2 Attention 模型架构

注意力机制的本质就是定位到感兴趣的信息，抑制无用信息，结果通常都是以概率图或者概率特征向量的形式展示，从原理上来说，主要分为**空间注意力模型，通道注意力模型，空间和通道混合注意力模型三种，这里不区分 soft 和 hard attention。**

2.1 空间注意力模型(spatial attention)

不是图像中所有的区域对任务的贡献都是同样重要的，只有任务相关的区域才是需要关心的，比如分类任务的主体，空间注意力模型就是寻找网络中最重要的部位进行处理。

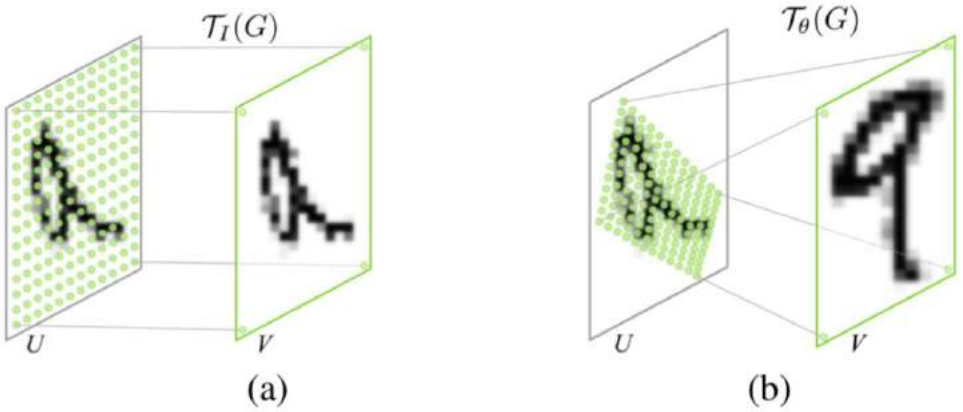
我们在这里给大家介绍两个具有代表性的模型，第一个就是 Google DeepMind 提出的 STN 网络(Spatial Transformer Network[1])。它通过学习输入的形变，从而完成适合任务的预处理操作，是一种基于空间的 Attention 模型，网络结构如下：



Spatial Transformer

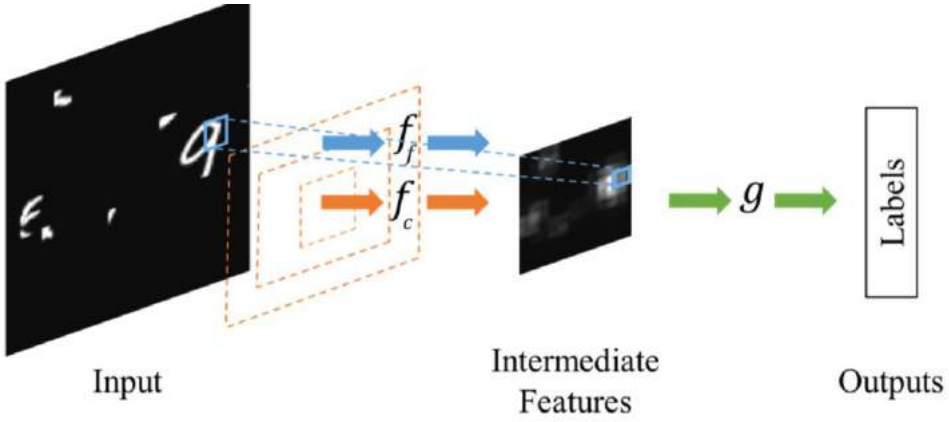
这里的 Localization Net 用于生成仿射变换系数，输入是 C×H×W 维的图像，输出是一个空间变换系数，它的大小根据要学习的变换类型而定，如果是仿射变换，则是一个 6 维向量。

这样的一个网络要完成的效果如下图：



即定位到目标的位置，然后进行旋转等操作，使得输入样本更加容易学习。这是一种一步调整的解决方案，当然还有很多迭代调整的方案，感兴趣可以去有三知识星球星球中阅读。

相比于 Spatial Transformer Networks 一步完成目标的定位和仿射变换调整，Dynamic Capacity Networks[2]则采用了两个子网络，分别是低性能的子网络(coarse model)和高性能的子网络(fine model)。低性能的子网络(coarse model)用于对全图进行处理，定位感兴趣区域，如下图中的操作 fc。高性能的子网络(fine model)则对感兴趣区域进行精细化处理，如下图的操作 ff。两者共同使用，可以获得更低的计算代价和更高的精度。

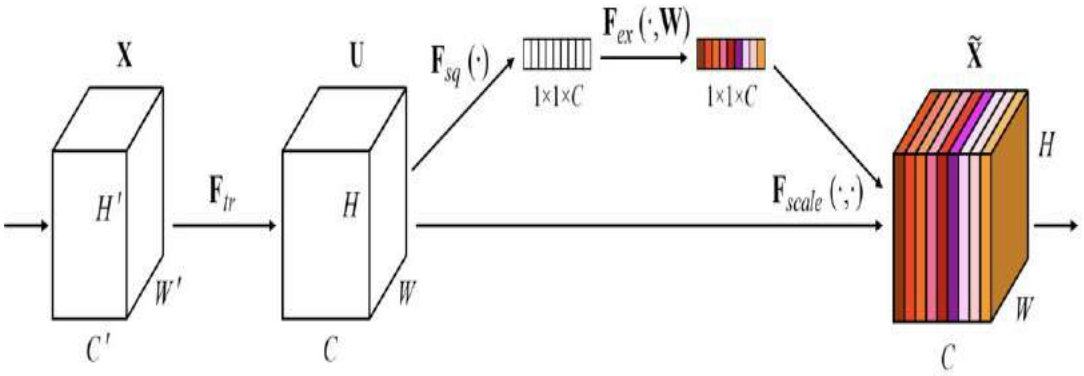


由于在大部分情况下我们感兴趣的区域只是图像中的一小部分，因此空间注意力的本质就是定位目标并进行一些变换或者获取权重。

2.2 通道注意力机制

对于输入 2 维图像的 CNN 来说，一个维度是图像的尺度空间，即长宽，另一个维度就是通道，因此基于通道的 Attention 也是很常用的机制。

SENet(Sequeeze and Excitation Net)[3]是 2017 届 ImageNet 分类比赛的冠军网络，本质上是一个基于通道的 Attention 模型，它通过建模各个特征通道的重要程度，然后针对不同的任务增强或者抑制不同的通道，原理图如下。



在正常的卷积操作后分出了一个旁路分支，首先进行 Squeeze 操作(即图中 Fsq(•))，它将空间维度进行特征压缩，即每个二维的特征图变成一个实数，相当于具有全局感受野的池化操作，特征通道数不变。

然后是 Excitation 操作(即图中的 Fex(•))，它通过参数 w 为每个特征通道生成权重，w 被学习用来显式地建模特征通道间的相关性。在文章中，使用了一个 2 层 bottleneck 结构(先降维再升维)的全连接层+Sigmoid 函数来实现。

得到了每一个特征通道的权重之后，就将该权重应用于原来的每个特征通道，基于特定的任务，就可以学习到不同通道的重要性。

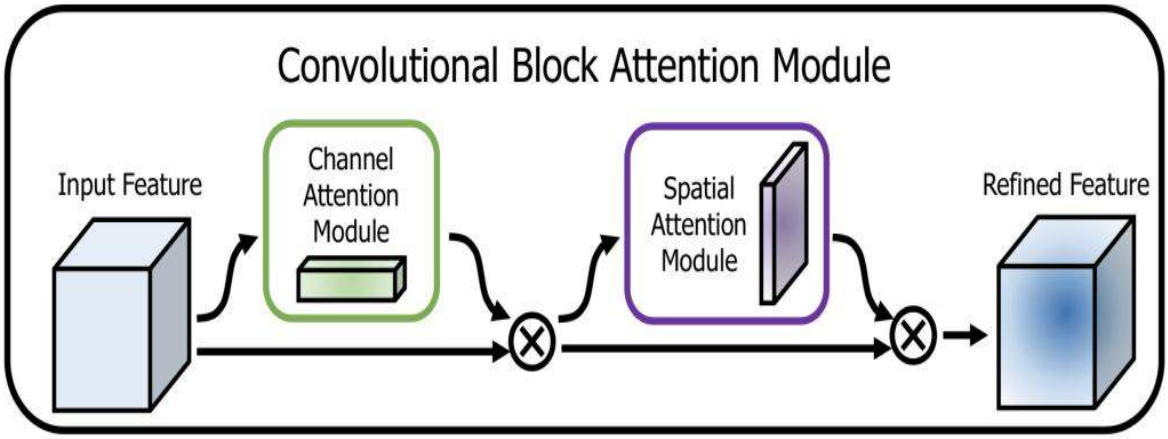
将其机制应用于若干基准模型，在增加少量计算量的情况下，获得了更明显的性能提升。作为一种通用的设计思想，它可以被用于任何现有网络，具有较强的实践意义。而后 SKNet[4]等方法将这样的通道加权的思想和 Inception 中的多分支网络结构进行结合，也实现了性能的提升。

通道注意力机制的本质，在于建模了各个特征之间的重要性，对于不同的任务可以根据输入进行特征分配，简单而有效。

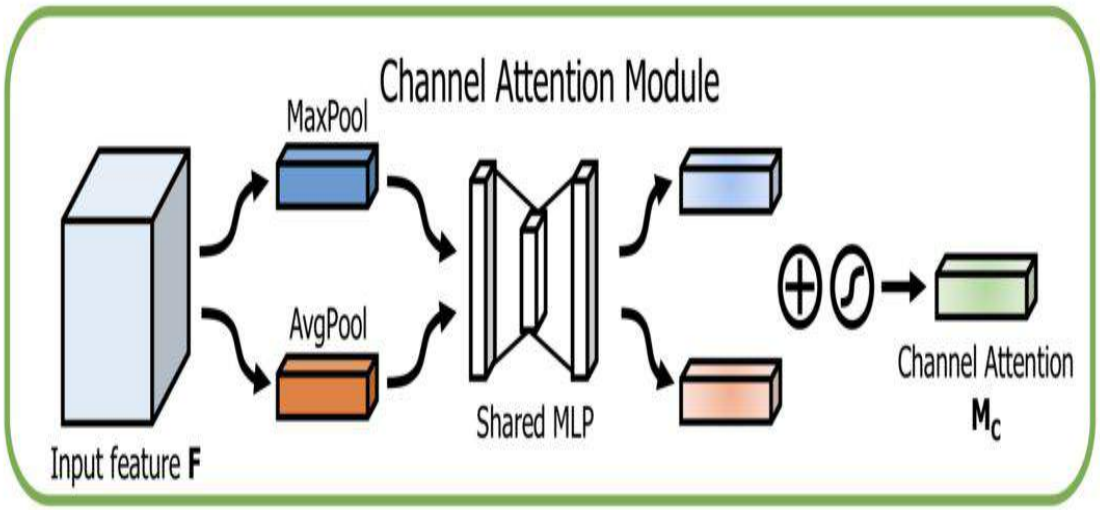
2.3 空间和通道注意力机制的融合

前述的 Dynamic Capacity Network 是从空间维度进行 Attention，SENet 是从通道维度进行 Attention，自然也可以同时使用空间 Attention 和通道 Attention 机制。

CBAM(Convolutional Block Attention Module)[5]是其中的代表性网络，结构如下：

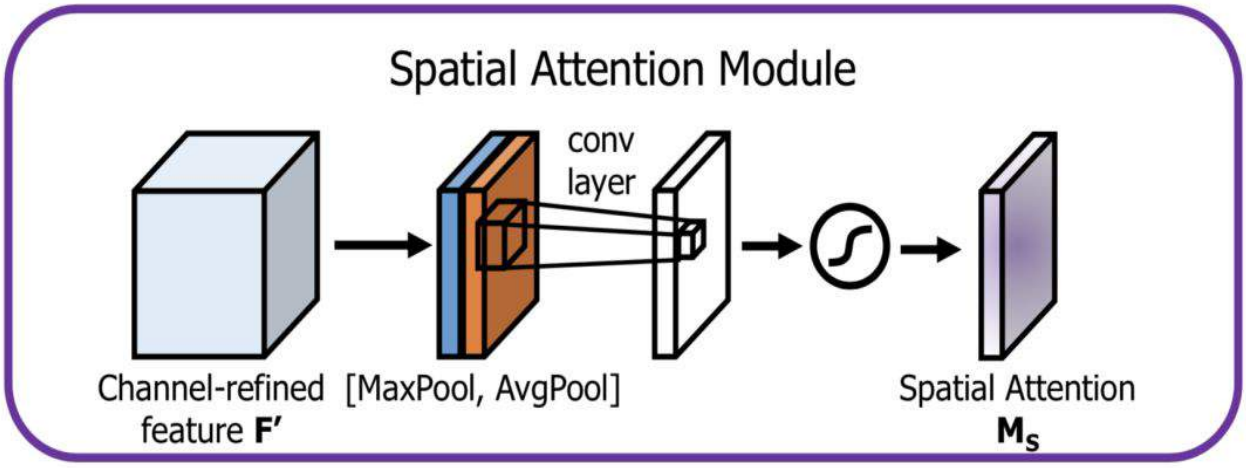


通道方向的 Attention 建模的是特征的重要性，结构如下：



同时使用最大 pooling 和均值 pooling 算法，然后经过几个 MLP 层获得变换结果，最后分别应用于两个通道，使用 sigmoid 函数得到通道的 attention 结果。

空间方向的 Attention 建模的是空间位置的重要性，结构如下：



首先将通道本身进行降维，分别获取最大池化和均值池化结果，然后拼接成一个特征图，再使用一个卷积层进行学习。

这两种机制，分别学习了通道的重要性和空间的重要性，还可以很容易地嵌入到任何已知的框架中。

除此之外，还有很多的注意力机制相关的研究，比如**残差注意力机制，多尺度注意力机制，递归注意力机制等。**

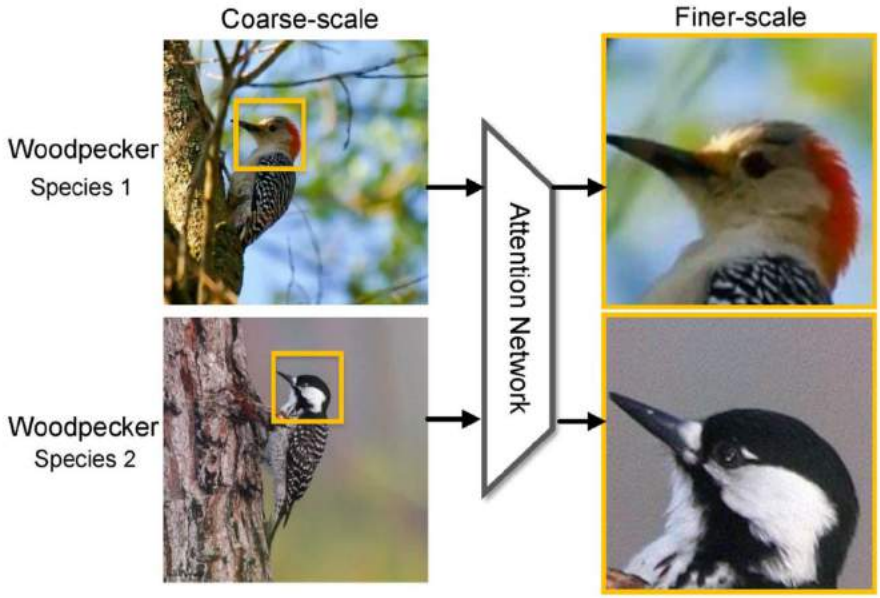
3 Attention 机制典型应用场景

从原理上来说，注意力机制在所有的计算机视觉任务中都能提升模型性能，但是有两类场景尤其受益。

3.1 细粒度分类

关于细粒度分类的基础内容，可以回顾公众号的往期文章。

【图像分类】细粒度图像分类是什么，有什么方法，发展的怎么样



我们知道细粒度分类任务中真正的难题在于如何定位到真正对任务有用的局部区域，如上示意图中的鸟的头部。Attention 机制恰巧原理上非常合适，文[1],[6]中都使用了注意力机制，对模型的提升效果很明显。

3.2 显著目标检测/缩略图生成/自动构图

我们又回到了开头，没错，Attention 的本质就是重要/显著区域定位，所以在目标检测领域是非常有用的。

上图展示了几个显著目标检测的结果，可以看出对于有显著目标的图，概率图非常聚焦于目标主体，在网络中添加注意力机制模块，可以进一步提升这一类任务的模型。

除此之外，在视频分析，看图说话等任务中也比较重要，相关内容将在有三 AI 知识星球中每日更新。

参考文献

[1] Jaderberg M, Simonyan K, Zisserman A. Spatial transformer networks[C]//Advances in neural information processing systems. 2015: 2017-2025.

[2] Almahairi A, Ballas N, Cooijmans T, et al. Dynamic capacity networks[C]//International Conference on Machine Learning. 2016: 2549-2558.

[3] Hu J, Shen L, Sun G. Squeeze-and-excitation networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 7132-7141.

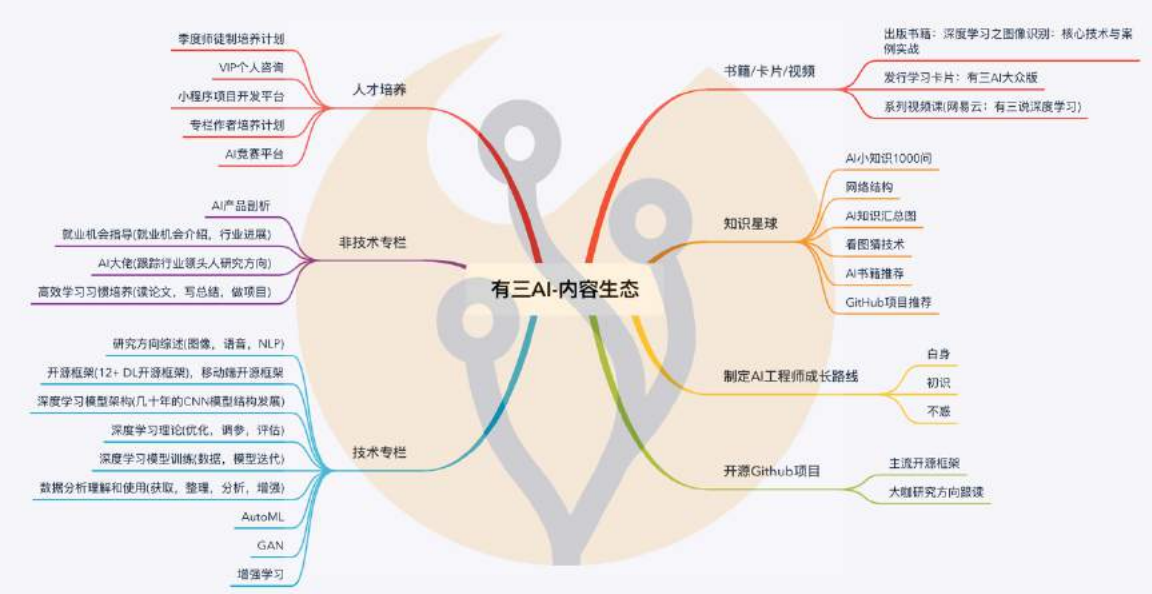
[4] Li X, Wang W, Hu X, et al. Selective Kernel Networks[J]. 2019.

[5] Woo S, Park J, Lee J Y, et al. Cbam: Convolutional block attention module[C]//Proceedings of the European Conference on Computer Vision (ECCV). 2018: 3-19.

[6] Fu J, Zheng H, Mei T. Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 4438-4446.

4 总结

注意力机制是一个原理简单而有效的机制，符合人眼的视觉感知原理，在实现上也容易嵌入当前的主流模型架构，是值得采用和学习的技术。



未完待续

有三 AI，2019 年 8 月 12 日