王垠精選文集

YINWANG.ORG

王垠文章愛好者整理
使用 TeXmacs 排版

目錄

學術篇

1 清华梦的粉碎——写给清华大学的退学申请

1.1 清华梦的诞生

小时候，妈妈给我一个梦。她指着一个大哥哥的照片对我说，这是爸爸的学生，他考上了清华大学，他是我们中学的骄傲。长大后，你也要进入清华大学读书，为我们家争光。我不知道清华是什么样子，但是我知道爱迪生和牛顿的故事。清华，大概就是可以把我造就成他们这种人的地方吧。我幼小的脑海里就想象出我能在清华做的事情.....我的脸上浮现出笑容。我说我要实现这个“清华梦”。这就是清华梦的诞生。

1.2 小小科学家

我相信每个人在小时候都跟我差不多，对这个世界充满了好奇。

鲁迅有他的百草园，我也有我自己的"实验田"。如果说小时候的鲁迅是一个艺术家，那么小时候的我就是一个科学家。 这么说可能有人要说我口气太大，张口闭口就是这家那家。然而在我的字典里，"艺术家"和"科学家"并不是什么了不起的人，它们只是贴在人内心的一个标签。 如果一个小孩专注于内心对世界的感觉， 那么他就是一个艺术家。而我不是。 我的大部分兴趣是在了解世界是怎样运转， 甚至不惜代价。 也许大部分男孩子都是这样。

我小时候住在父母执教的中学里。两间平房，门口有一小块地，妈妈在里面种了一些菜。 我们一家三口虽然穷，但是过着宁静舒适的生活。 我们在这个地方一直住到上初中的时候。这些房屋记录着一个年幼的科学家的探索和实验，直到它们被夷为平地。

妈妈拒绝让我养猫狗，她说凡是会拉屎的都不养---除了我。 所以我小时候就喜欢与蚂蚁作伴。 我总是试图用各种各样的办法去了解蚂蚁的生活习性。 我可以一整天的观察我家屋檐下的蚂蚁来来去去。 看见他们用触须碰一碰， 然后各自分头走开， 我就会想它们到底说了什么。 我在想，能不能用一种方法解开蚂蚁语言的密码。 我从书中得知蚂蚁洞里有蚁后，她有很大的肚子。 为了一睹芳容，我开始试图水漫金山，把水往蚂蚁洞里灌。 我有时一个下午就干这种事情，却没有一次成功看到蚁后。 后来才知道蚂蚁是如此精明的下水道工程师，水大部分都渗到地底下去了。 可是我不甘心，我开始试用别的办法。比如在洞口放一块糖。可是蚁后架子太大，终究不肯出来，让别人帮她送饭进去。

有人说，这个世界最后不是毁在疯子手上，就是毁在科学家手上。世界上如果只有科学家是很可怕的，比如他们会发明高效的杀人武器。我发现疏松的棉絮可以迅速的燃烧，就想出一种惨绝蚁寰的大屠杀实验。我先把糖水滴在地上，等蚂蚁把那个地方围个水泄不通的时候，铺上棉花，点火.....现在想起那些勤劳的小黑头都变成灰烬，我仍然心惊肉跳。他们的灵魂会来找我报复吗？后来这个实验有一个升级的版本用的是浸泡过一种化学药品溶液的纸，文火燃烧，由于燃烧速度慢，杀伤力不大，这个实验可以测试蚂蚁的逃跑路线。我还用活蚂蚁进行过心理实验。首先用破袜子摩擦塑料尺产生静电，然后放在一只正在行走的蚂蚁身后不远处。蚂蚁走不动了，我就开始推测它在想什么，它感觉到什么。它可能会觉得有外星人？但是由于尺子拿开以后，它若无其事继续走，我猜它只是有点纳闷，而不惊慌。但是反反复复几次之后，它明显有罢工的意思，似乎忘了自己要去干什么。后来我又发现蚂蚁被吸到塑料尺上之后会由于带上相同的电荷而被"发射"出去，就像人间大炮一样。注："人间大炮"是日本电视剧《恐龙特急克塞号》里的一种可以把人当作炮弹发射的威力很大的电磁装置。

一点微小的发现，就可以引发大量的探索和实验。这就是我在那个年代的特点。虽然妈妈也逼着我练习书法，绘画，还多次获奖，但我不喜欢这些东西。我似乎生下来就是科学家，不是搞艺术的，不过也许只是妈妈的强迫让我反感了艺术而已。物理是我最喜欢的，因为它让我了解到世界的奥秘。我一般开学前几天就会把物理书上的实验都挑出来，费尽辛苦找到材料实践一番，心里美滋滋的。上学真是快乐！

1.3 失之交臂

上了高中，由于课业的压力，我的生活逐渐改变了。为了考上清华大学，我努力的学习。抛下我的毛笔，抛下我用来做实验的蚂蚁，电池和线圈，抛下除了考试科目的一切。在老师眼里我是一个听话的好学生，在妈妈眼里我是一个听话的好孩子。每天早上按时起床，吃一大碗妈妈做的面（为了补充一上午学习需要的体力），然后冲进教室，按照预设的程序开始读书，做练习题。似乎一切都有条不紊，顺利进行。可是.....

忽然有一天我发现，我的一切活动都是在纸上进行的，看书，做习题。试卷和复习书让我变得变得麻木。我想这样下去我就不再像爱迪生和牛顿了。于是我开始调皮起来。我不但要做考试的题目，还要做更难的题目。做了物理奥林匹克的题目，接着就想看大学的物理书，接着就想恢复我小时候的实验的爱好。老师辅导自习时经常被我缠住问一些不着边际的问题，那其实是我在实验中发现的问题。终于有一天，在我要求他跟我合作制造一个磁悬浮陀螺的时候，他显示出了不耐烦："王垠，你让我先回答别的同学的问题好不好？你的问题对考试没有好处。"我呆住了，启发我让我爱上物理的人，尽然对我说出这样的话。后来想一想，他也是无奈啊，不过我从此再也不想问他任何"超纲"的问题。

高二的时候妈妈就拿回一份前一届的高考题让我做，我随手一做就得了一个当时可以考上清华的成绩。我的心里想，清华我来了。明年的这个时候，我就会拿到录取通知书了！从此我就不再把高考放在眼里。我开始钻研越来越难的题目，进行越来越离谱的实验。我想，清华里面应该都是我这样的学生吧，我会有很多志同道合的朋友，不用再跟这群只会做题的呆子在一起了。

可是我的行为总是受到老师的压制，他们要把我们变成考试的机器。他们告诉我，沉下心来做习题，考试才能有把握。妈妈也帮着老师劝导我。看，一班的某某某这次模拟考试数学成绩比你高，多努力一下吧。我哪里听得进去，我才不在乎这点分数，我能解决更难的问题，老师都没法解决的问题。我开始有了逆反心理，开始早上懒床，装病请假不去上课。班主任，校长多次找我谈话，说我要沉下心来准备考试云云。但是我根

本就听不进去，我鄙视高考，觉得他们没有资格出题来考我。然后我就有了心理疾病，大概是强迫症。高考语文的时候我居然怀疑监考老师认为自己在作弊，接着好像真的怕被抓住了一样，手发抖，头冒汗。然后我又想要是考不好，以前的优秀会不会也被人怀疑？他们会不会以为我以前的成绩全都是作弊得来的？手就抖得更厉害了。这时候，监考老师可能发现了我的情况，真的走了过来，站在我身后。害得我好几分钟不敢写一个字，因为手已经完全不听使唤。不过他还是走开了，这可怕的高考终于结束了。

我们是考试前填的志愿，我根本不考虑其他学校就只填了清华。后来妈妈研究了一下，帮我添了一个天津大学在第二志愿。以下的志愿全部空白。大家觉得我真够大胆，可是我的心理状态让我发挥完全失常，比清华的最低分数线还差两分。特别是语文，才 96 分。天津大学第一志愿收满不要我。昔日的好学生，居然到了落榜的下场。我真的那么好吗？我问自己。我太骄傲，才落到如此地步吧。我开始怀疑自己是否应该那样瞧不起高考。看着爸爸的愁眉苦脸，妈妈的唠唠叨叨，真是生不如死。复读吗？那会是噩梦的继续。我不能再在这个学校待下去。再面对题海，我的心理疾病会让我自杀的。碰巧四川大学来招收高分落榜的学生，还给了我随便选择专业的机会。妈妈说，计算机现在很火热，出来好找工作。我虽然对工作不感兴趣，但是我比较喜欢写程序，于是就进了川大计算机系。

1.4 两度退学失败

不能不说进川大是个没有选择中的好选择。大学生活自由一些，我至少不会走上自杀的道路。可是我的毛病仍然在继续，我永远不满足学校里能学到的那么点东西。老师基本是照本宣科，我逐渐不再满足这种知识灌输式的教育。我觉得完全没必要上这个大学。

川大的环境我实在无法忍受。军训的时候受够了同学和教官的委屈，我就想退学。我们的军训是在一个戒备森严的炮兵基地里，心里的苦向谁说啊！有一天我们正在路上齐步走的时候，我忽然看到一个女人挽着一个军官走了过来。那个军官的老婆怎么长的这么像我妈妈！要是妈妈来到我身边该多好！没想到回到营地，团长（原来是连长，我们来军训他就升一级做团长了）说有人来探访。我走过去，居然发现是妈妈！因为听说我想退学，她急忙向学校打听了军训的地点，几经周折跑过来，是那个军官带着她混进来的。我想我妈妈要是转行当间谍一定是个好料子。她说已经帮我办了退学，学校同意了，回去好好复习，准备考上清华…… “好好复习，好好复习”……我的脑海里又浮现出高三的情景，这次我要跟一群更没用的复读的人在一起。脑子一阵疼痛之后，我说：“妈妈，我不想退学了。”

可是军训回到学校，发现宿舍如此差劲，我又想退学。妈妈又来帮我办理手续，可是结果我还是由于懦弱反悔了。害得学校办事的老师都骂我：“你这个人简直神经病！”对啊，我确实是有病，不过我的是精神病，不是神经病。我恨我的高中，我恨我的大学，我恨高考，我恨中国的教育！是你们让我生病的。可是妈妈，她为了我已经费尽了辛苦。我不能再这样周折下去。我自己在学校里好好努力，准备考上清华的研究生吧。

学校住宿环境很差劲，又经过好多麻烦事，我终于决定在校外去租房子住。后来我开始玩滑板，它让我变得勇敢。我心里逐渐平静下来，可以用心看书了。大二以后，我的学习生活才逐渐进入正常，自信开始恢复。

1.5 梦的复苏

记得川大教 Pascal 语言的老师第一堂课就对我们说：“我们学校就是落后啊。外面公司里都用 C, C++ 了，我们还在教 Pascal。你们以后要出去工作恐怕还是得学学 VC 什么的。” 于是有的同学开始抱起一本本像 “XXX 圣经” 之类的书开始学习，上数学课也在看这些东西。我当时自愧不如啊。自己就是小学的时候玩过一下学习机，可以说没有任何计算机基础。辅导员也经常夸他们几个动手能力强，以后公司就需要这样的人。他们出口就是 Bill Gates, 世界首富云云。军训的时候听着他们说什么 DOS, 温 95，我就只有张着嘴崇拜的份了。才想起我高中计算机竞赛的时候一道有关 DOS 命令的题没有做出来，现在听他们说才知道原来 DOS 是个“操作系统"。那操作系统又是什么，他们说每个电脑上都必须有一个操作系统.....我真是愧不如人 -_-!

正在我决定鼓起勇气后来跟上，准备拿起一本 DOS 大全从头啃起的时候，一次偶然的机会我接触到了 Linux. 后来又因为 The Art of Computer Programming, 接触到了 Knuth. 我才发现，课堂上讲的那些东西原来如此低级，有些东西学了就过时，学它干吗？我并不比别人落后多少。我一再的思考，什么是计算机科学？是什么让我们计算机系的人不同于其他系的。我有时候认为有了答案，但是后来答案又被我自己推翻。在思想的混乱中，我发现我逐渐摆脱了旁人的标准。我不再想像别人那样去考计算机等级考试，对微软的认证也不屑一顾。我自己学会了 Linux, 还会很多种当时别人听都没听说过的计算机语言。我学会了 LaTeX, 还因为找出 Knuth 书里的错误得到两张支票和一些礼物。这并不是什么值得炫耀的，但是这给我对计算机的兴趣很大鼓舞，我从此更加认真的看书。上课要不就逃掉，背地里拿着大部头的“龙书”之类的原版英语书啃。要不就看我打印出来的 GNU 的一些资料，完全不听老师讲。期末划重点的时候也不去，考试却总能考个八九十分。总有几个女生排名在我上面，不过我不在乎这点分数，考试和分数不再能评价我。同学们大概都觉得我是一个怪人，后来毕业了我才听他们说，他们管我叫“怪才”。我如此努力的学习着，对别的事情充耳不闻。我只有一个目的，就是毕业就离开这个鬼地方，进入清华大学上研究生。虽然大家不理解我在干什么，清华的老师应该挺在乎我学的东西吧。

可是我没有想到，在我死啃书本的时候，我的创造力正在离我远去。在我盲目接受我认为高深的材料的时候，我失去了自己的创造。我成了比别人稍微好一点的技术工人，不再跟爱迪生和牛顿是一类人了。我高中的时候拼命想保存的创造力已经在苦读之下消失殆尽。我看书的方式变得顺序化，总想从头看到尾。我的高中老师们的目的，中国教育的目的，终于快达到了。

1.6 清华，我来了

大三的暑假，我来到清华想拿一些考研的资料。这是我梦中的地方呀，美丽的校园，比川大要大上好多倍吧，脚都走痛了才走到招待所。去系办，一个办事员态度很不好的给我一份资料。哎，学校好，人脾气就该大啊。忍了吧，要是真能考进来就好了。

后来听一个老师说清华有一种学生叫“直博”，可以硕博连读，五年拿到博士学位。只要面试通过就可以进来学习。我心想这种方式好啊，我平生最讨厌的就是考试了。出高考题的那帮人，他们有什么资格考我！考研资料也是遍地飞。写了几本复习材料就自称什么“一代名师”，我最看不起这种人了，就会赚钱。我如果可以获得“直博”的名额，就可以永远摆脱他们了。想一想，要是硕士三年，博士三年，就要六年。现在五年就可以拿到博士学位，还不用考试，真是太好了。可是我又有什么资格获得清华的直博？我在川大从来没听说过这种东西。

于是我就开始打电话联系老师，跟他们谈谈。面对他们的眉头，面对他们的笑脸却无可奈何的说“没有名额”，我都感觉没什么希望了。一个院士甚至对我说：“你们四川大学是什么学校？二流都算不上，最多算个三流大学。你怎么能来我这里！”我深受打击，可是我还是没有放弃。最后我找到了一个老师，我们一开始就谈的挺投机。他听说我跟 Knuth 有过联系，挺高兴的说，哦我知道他，好多年前来我们这里做过报告呢。我终于觉得找到了知音，于是决定就跟着他学习。老师找好了之后还有一个面试，是别的老师参加的，我说什么他们似乎没有认真听，就一个劲看我的考试成绩这种我不屑一顾的东西。我面试时特意穿上了 Knuth 送我的 MMIX T-shirt, 他们大概根本不知道是什么，我也没有提起。

不过老师只对我的体育成绩提出了疑问，说你怎么才 80 多分？你的身体能不能胜任繁重的学习任务啊？我笑着回答，我每天还跑 5000 米呢，我们学校打分比较严，难道清华的学生体育都考 90？面试就这样通过了。

1.7 推荐信与散伙饭

面试通过后回到学校还要办一些手续。成绩单，推荐信等等，跟申请外国大学研究生院差不多，让我感觉挺正规的。院长对我挺好的，同意帮我签推荐信。可是签完字之后他对我说：“你别以为他们觉得你是个人才。他们是根本招不到人！他们那里像你这样的学生都出国了，剩下的是最差的。谁想读博士啊？你别太高兴了。”我笑着应付这突如其来的打击，在心里却不断为自己的选择辩护。清华一定是好样的，不会让我失望。它是我的梦啊。

很多麻烦的手续之后，终于拿到了我梦想的大学的录取通知书，可以离开川大这个鬼地方了。毕业的散伙饭上，看着大家喝得酩酊大醉，还有人在咆哮说居然连川大的文凭都没拿到，我一个人默默地想象着即将到来的清华的快乐生活，暗自庆幸。

散伙饭到了尾声的时候，我诧异的看到一个平时不太熟悉的同学拿着一杯啤酒走过来。我挺紧张，我最不喜欢别人给我敬酒了，说是客气，其实很虚伪。没想到他说：“我敬你一杯，大牛人。听说你被清华大学录取作了博士。我干了，你随意。”我不知如何回答，我一向不知如何应付别人的恭维。还好他没有让我也干杯，倒是够尊重人。没想到喝完他接着说：“我知道你是怎样的人。我很仰慕你，你是真正喜欢研究的人。可是我要告诉你，清华的人并不会比我们好多少。大部分人也只是想混一个学位，将来找个好工作。没有多少人可以跟你一起研究的，你去了必定很孤独。我就很奇怪你这样的人怎么不出国呢！你会后悔的。”

我有点不高兴了。一个人说你的选择是错误的，你的反应是什么呢？反正我当时为我的“清华梦”作了一番辩护，说我进去自己好好研究，应该还是能够很好的，毕竟这是我从小的梦啊。可是没想到，他说的居然是对的，我现在开始感谢他了。

1.8 计算几何，创造力的复苏

清华还是一样的上课方式，大部分课也是很多人一起上，一起打瞌睡。老师也是照本宣科，我居然发现他们其实跟川大的老师没什么区别。清华的不同之处就是，一到考试的时候原来进行的一切娱乐活动都不见了人影。原本每天晚上都有人一起玩轮滑，考试的时候就只剩下我孤零零的一个人。因为大家都怕考试，开始熬夜复习了。还有就是上课不容易逃课了，有些老师会突然点名，缺席会严重影响最后的成绩。

对于博士生，传说还有一个规定，那就是后 10% 淘汰。也就是说，不管你成绩如何，如果成绩排名在课程的后 10%, 那么就要重修。而如果两门功课重修，就会被开除。面对如此残酷的规定，很多同学都惶惶不可终日。我就是在隔壁同学的唠叨声中度过了第一期。不过我还是没有把考试当回事，所以我也没有去验证这个说法的官方真实性。我仍然不去听老师划重点，我仍然不觉得老师出的题目有什么好，我仍然讨厌有人让我们用手算矩阵。可能觉得太残酷，还是觉得要是开除了博士生谁来干活，这条规定后来改成了如果博士生上了 80 分就可以不重修。我也不知道为什么我觉得考砸的科目也上的了 80, 故意放我过去的吗？

但是我的生命中出现了这样一门课程。它改变了我对老师的看法，让我觉得上课原来也可以如此有趣。这就是计算几何。上课的人很少，只有十来个人。因为听说这门课很难，很多同学都没有选。但是我就是那种知难而进的人。老师上课的方式跟别的课程很不一样，大家坐在一个小教室里，老师有精美的幻灯片，有动画，不时还插入一段大科学家，大哲学家的名言。上课时老师会停下来很多次让学生提问题，下课大家都积极踊跃的讨论新奇的问题。课程的评分方法也很特别，平时成绩占到 30% 的分量，作业分为几种分值，可以自己选择做不做，作业的总分数乘以 30%, 加上最后大作业的分数乘以 70%, 就是最后的得分。说真的，这门课太有趣了，我就只逃过一次课。但是还是有时候人数不到一半，因为其他课程压力太大，有人都去复习别的课程了。但是邓老师从来不点名，还对逃课的同学表示同情。还问我们在座的有没有其他课特别紧张的，下次课可以不来。真是让人感动。

我就是在这门课上认识了王益，我们亲密无间的合作，让我领略到了什么叫做研究。大作业的时候我们在一个小组，其实是三人一组，但是那第三个人其实什么也没干。我和王益决定写一个 3D 的 Voronoi 图扫描算法演示程序。王益的 3D 图形编程能力很强，所以他做界面，由我负责算法生成数据作为后端。我们分别在自己的机器上编写程序，不时的打电话讨论接口的设计问题。我找到了 Bell labs 的 Steven Fortune 的算法程序，决定看懂它，然后改造成演示需要的分部运行的算法。但是 Fortune 的程序几乎没有注释，而且使用了一种奇怪的数据结构，很难理解。Fortune 还在程序里说到，这个算法虽然有效，但是对于程序员来说是一个挑战。所以我 email 请他给我一份算法论文的拷贝，他同意了。但是一个月之后，信才到我手里，那时我们已经完成了作业。因为我花了一个星期看懂了他的程序，还换掉了他的麻烦又低效的数据结构。随后成功的把后端与王益的前端设计好接口联合。等我看到 Fortune 的论文，发现程序里面其实已经改进了论文的核心内容。其中的 parabolic transformation 其实完全没有必要实现。我深深体会到实践的重要性，也许先有了他的论文我反而会被误导，写不出实际可以运行的程序。

由于我们的团结努力，老师对我们的大作业非常满意，他给了我们最高的分数 100. 由于我们两个都在课下超额完成作业，所以总的分数我们两个都是满分。这是我阔别已久的 100 分。只有在小学我才拿到过这种分数啊！对于一个对考试成绩满不在乎的人，100 又意味着什么？如果是别的课程我会毫不在乎，就像我得了 80 分一样。可是这个 100 分是我们团结研究而来的，它包含了对我们的合作意识，对我们的友谊，对我们的热情的肯定。虽然我觉得我们的东西还有改进的余地，但是我接受这个 100 分！也只有这样的课程，我才可能得 100 分。

从此我感觉到了什么叫做研究。这跟我小时候干的那些事情没有什么两样。你在身边发现一个问题，想知道为什么。然后你就想去获得解决这个问题的知识。你去看书，你去问专家，你上网去搜索。如果没有发现答案，那么好啦，你就可以自己试图去发现为什么，这是最有趣的部分。知道了为什么，就想让这个东西有用处，对人们的生活产生好处。这就是研究。

1.9 《完全用 Linux 工作》与 TeX 的推广

这么说来我还是对清华有些好感。遇到一个好老师让我从呆头呆脑的技术工人的状态恢复过来，开始追求自己的梦想。可是第一年把所有的课程上完之后，我就发现原来清华所谓的"研究"是如此混沌。其实清华大部分人进行的所谓的"研究"是什么呢？其实就是写作，不是科学研究。这一点以后我会详细叙述。

远远看去外观华丽的有着先进的工作站的实验室，却没有可以安心看书的地方。机器挨着机器人挨着人，书都没地方放。师兄师姐们都在忙着用 word 写论文，不时有两个人隔着几行机器大声谈话。实验室通风不好，还有一个大型工作站在嗡嗡作响，我进去一会儿就觉得头晕，所以后来就不想去了。PC 机以前都是公用的，每次都会用不同的机器，却没有我想用的软件，麻烦死了。好不容易实验室买了新机器分配给个人，装上一个 Linux 系统开始写程序，还在 Sun 工作站上安装了多达 1G 的 GNU 程序。却被一个师兄嘲笑说那种跟 DOS 一样落后的东西你居然也用。于是我写了一篇文章叫做《完全用 Linux 工作》，放在主页上驳斥这种观点。矫枉过正，确实写的优点偏激，结果引起网上 linux 界轩然大波。后来我又发现几乎全校的论文都是 word 排版的，那些公式质量太差，看起来头痛，才发现很多学生害怕数学的原因之一。所以又写了文章宣传 TeX, 希望中国产生更多漂亮的数学书。这下子我出名了，真没想到，出名不是因为我的研究成果，而是因为这些业余的东西。我起初不希望我因此出名，但是看到旁边的人都用上了 TeX, 我觉得我还是做了一件好事，至少让论文看起来漂亮了一些。

可是论文的内容，却是我永远的痛！

1.10 培养计划

我在第一年就把功课全部上完了。本来我想多选几门课，比如法语，可是清华的博士要选课需要提交一个"培养计划"给导师签字。导师同意之后才能修改。导师看到我选了法语，就说这个第二外语还是自己学学就行了吧，旁听也行啊，我主要是怕你课太多了考试不通过就麻烦了。我当时没有说什么，就把法语去掉了，只留下刚够学分的课程。其实我还想选很多的，体育，音乐什么的，都不好意思跟导师说。后来才知道宿舍对门的硕士生选了钢琴课他们导师都不管。为什么我们就受到如此待遇？

可是没有把法语加到培养计划却成了我的遗憾。有一个新学期我去旁听了第一节法语课之后老师就说，我知道很多同学是来旁听的，这样教室里人太多了，效果不好。这对自己对大家都不好，下次请旁听的同学不要来了。我脸皮薄，下次就没有去了。后来自己想自学却又没有老师教，看了十集 reflet 之后就此作罢。

后来我终于明白了，清华不需要全面发展的博士生，而其实导师还会在某种程度上削弱学生的能力。导师并不是真的为我们好，而是不喜欢我们上课，因为上课不但会花掉研究（或者干活）的时间，而且让他们眼界太开阔，这样学生会很容易有别的选择而走掉。所有的活动：助教，实习，都必须有导师签字。而大部分导师就会找借口不让学生干这些事情。不给他们助教和实习的机会，让他们以后不好找工作，只能为自己服务，或者为自己的熟人服务。甚至这次我去西藏，要办边境证都要有导师签字。办事的老师说，没有导师签字，你跑出去了不回来怎么办？大妈，我跑那种地方干吗？

除了这些，还有两大法宝就是博士学位和违约金。清华的博士学位有多值钱知道吗？不知道？那么博士退学要交几万块钱的违约金，这下大部分穷苦学生怕了吧。这就是你们的卖身契。清华就是这样把研究生牢牢地控制在自己的掌握之下。我对一个如此害怕学生跑掉的不自信的学校还能说些什么？如果你是好样的，就不会害怕我们跑掉！该跑掉的最后终究会跑掉。

1.11 我的自我培养

在学习上，我永远是个吃不饱的人。选不了课，我就去旁听。旁听后觉得老师讲的不好，我就自学。在我有空的时候，我就会去图书馆借书看。在我本科的时候，我就已经发现自己的一个特点，我会很快发现新的东西，并且学会使用它。虽然这些东西并不是创新，但是它们丰富了我的技能，让我有更大的能力去进行创新。我经常顺藤摸瓜似的从一个问题搜索出一大串我想知道的东西。然后借一大堆书回来，每本看一点点，只为找到我需要的答案。

计算几何课的一次作业，我为了写一个算法的演示程序，花了 3 天时间学了一点 Java 语言，正好能够完成那个程序。我开始接触到 TeX 的底层细节，看完了 The TeXbook, 并且找出一道练习题答案的错误。开始移植 gbkfonts 程序，作为我的 CWEB 语言的练习。看完了几乎所有 Xlib 的手册，了解了 XWindow 的工作原理。我接触到 Scheme，并且做完了 SICP 的大部分习题，还自己想出好多问题用 Scheme 实现算法。后来花了好几个晚上，把 MIT 课程 6.001 的录像下载回来。我才发现教授上课可以如此搞笑有趣，上课时戴上巫师的帽子，做一些滑稽的表演。我终于明白，有的计算机科学家居然可以去好莱坞演电影 :) 这个课程让我领会到 LISP 的强大，改变了多年以来对这种古老语言的误解。它让我感觉到在看似纷繁复杂，不断更新的计算机语言的世界，还有那么一种永恒的美！接着我又学会了 Common LISP, 并且开始用它来设计研究计算几何的一个函数库。另外还找了一些希奇古怪的程序来玩，写了一些心得体会放在网上给别人看。

我意识到自己数学还不够强，甚至有些怕，就开始看一些数学方面的书。Concrete Mathemtatics, What is Mathematics?, Science and Hypothesis, Godel Escher Bach, ... 虽然每一本都没有看完，但是我逐渐相信自己的数学能力，发现数学原来如此有趣，并不是做习题那么枯燥，也不像一辈子就拼命证明一个定理那么清高。才发现国内很多数学书用难看的符号把学生吓倒了，其实想通了就是很直观的原理。

我看了电影 A Beautiful Mind 之后深受感动，就去买了一本原著的书，它是数学天才 John Nash 的传记。它描写了 20 世纪初的 Princeton, 一群科学家生活的情景。我眼前浮现出在一个房间里，一群人在喝茶聊天下棋讨论问题激烈争论。我发现我从小内心向往的，就是那样的地方。我看到 Nash 是如何用“头脑暴力”解决一个他没有任何基础知识的问题。原来只要有了问题和探索的精神，就会有动力去获得解决它所需要的知识，最后将问题解决。发现有用的，重要的问题，而不只是寻找困难的问题，这样才会对人类有价值，才会有动力。我还看到一个真正的数学天才是怎样的喜欢恶作剧，又怎样因为过度的傲慢狂妄，想向世人证明自己的天才而发疯。我发现世界上有远比科学更宝贵的东西。我开始悔悟我高中时对待成绩不好的同学的态度。我不是一个天才，但是我要做一个好人。

但是我的研究却没有多少进展，至少我自己这么认为。我发现问题的根源，就是没有真正的讨论，没有真正的问题。

1.12 我们也有讨论，原来是这个样子

上完课，就该开始搞研究啦。可是研究什么呢？老师给我几篇论文看，意思是让我看看有没有什么想法。

我开始感觉没有头绪，就跟导师说能不能找师兄师姐跟我讨论讨论，还有别的人在做这个吗？他说，就你一个人做这个，每个人做一个题目，独立思考，这就是研究。我觉得是啊，我应该独立思考。可是过了一段时间发现不行啊，我想实现一个想法，但是我不知道是不是已经有人试过失败了。实验的时间开销会比较多，所以我想知道那么多厉害的人，为什么都不用这种明摆在那的方法？当我再次提出需要讨论的时候，他似乎有点生气的说："你为什么总是想有人跟你做一样的东西啊？你不是想抄袭别人的论文吧？"我不发话了。继续做我的实验，结果确实不理想。虽然自己实践很重要，可是要是能利用别人的经验，何乐而不为呢？这并不是偷懒。如果有人讨论，很多时候一个人提出一个问题，另外的人可能就会告诉他这个问题是不是有人做过，有什么重要性，凭直觉告诉他有什么难度。可是如果没有讨论，连问问"有没有人做过"的机会都没有！

后来我就经常上网看看国外的大学怎么搞研究，发现他们都有 seminar, 讨论组。A Beautiful Mind 描述的 Princeton 以前的天才们每天都在一个地方喝茶，讨论问题，争得面红耳赤，回家分头思考，做实验，第二天喝茶时再讨论。那就是我从小梦寐以求的生活啊！计算几何课已经让我爱上了与人合作和讨论的方式，现在却孤零零一个人了。我必须告诉导师，合作和讨论是非常重要的。在我据理陈述之后，他说："好吧。反正师兄师姐各自有自己的事，你要讨论什么就跟我和你副导师讨论吧。"于是我就开始了跟他们两个星期一次的见面讨论。每次讨论都感觉他们不知道我在说什么，他们心里想的都只是这个能比别人的好多少呢？能不能投到这个会议呢？如此宏观。我觉得跟他们讨论完全是浪费时间。

后来课题逐渐有了新的同学加入，导师决定跟中科院数学所的人一起申请一个项目来研究。于是我们每两个星期去中科院讨论。不过感觉他们那边也差不多。中科院的老师觉得他们的研究太理论，期望我们能给他们带去一点实际的东西。可是我们也没有什么实际的东西，所有的问题都是从别人的 paper 里看到的。副导师就开始跟他们说这个问题有多么多么重要..... 他们也借此机会开始研究以前放下的一些问题。总之讨论的感觉就是没有目的，没有主题。有时有人说他在想一个什么问题，说了一会儿就被否决了。有时候就是一个人看了一篇 paper 之后做一个感想。我坐在那里就在想，我们到底在干什么？我们甚至都不知道什么东西值得研究，还研究什么？后来师弟师妹们就开始考虑把问题变一变，看看能不能产生新的问题。他们的做法，我跟他们开玩笑说就是"有问题也要解决；没有问题，制造问题也要解决！"他们笑着点点头，"本来就是这样嘛。没办法啊。"

博士生论坛的时候，同学们都觉得有类似的问题，讨论不足，交流不足。所以我提议成立一个类似国外大学的 Common Room, 用来讨论问题。可是大部分老师说："这样一个房间，天天都要有那么多人在里面待着。谁来出这个钱？"是啊，老师自己的办公室都要钱，哪里可能有什么 Common Room? 就算有了 Common Room, 在里面讨论的无非还是文章发到哪里的问题。制度决定了行为，我的设想太理想化了。

博士生论坛的时候，同学们都觉得有类似的问题，讨论不足，交流不足。所以我提议成立一个类似国外大学的 Common Room，用来讨论问题。可是大部分老师说："这样一个房间，天天都要有那么多人在里面待着。谁来出这个钱？"是啊，老师自己的办公室都要钱，哪里可能有什么 Common Room？就算有了 Common Room，在里面讨论的无非还是文章发到哪里的问题。制度决定了行为，我的设想太理想化了。

分析一下，为什么老师不提倡讨论呢？因为问题是有限的。老师辛辛苦苦这么多年搞来搞去都在搞这些问题，分配给你们每人一个，互不冲突。要是两个人都搞一个问题，这下好了。出了成果论文归谁？学校要求必须第一作者才算论文数。要是两个人都写论文，那么投到同一个会议肯定有一个要被 reject. 这样对集体发展不利嘛，大家不就是发几篇论文混毕业吗？何苦？

1.13 paper, paper, 还是 paper

说到 paper 我就痛心。我的方向上我至今还没有看到几篇我觉得像样的文章。我主要进行集成电路布线算法的研究。看起来高深，其实是很简单的问题，一个平面上有一些点是电路里的电极，现在需要用铜线把它们连起来，怎么样让连线的长度或者时延最短？这个问题跟几何上一个有名的问题 Steiner tree 问题有关系。我的导师就是以前写了一篇这样的 paper 发到 IEEE transactions.

已经毕业的一个师兄就在他研究的基础上修改来修改去，发了好几篇 paper. 英文的不够还翻译成中文，投到国内的期刊。后来一个师姐又在这个师兄的基础上进行修改，又发了好多篇。可是在我看来，他们的论文纯粹就是炒冷饭，没有什么创新。一个问题解决了，那么解决问题的人显示了他们的聪明，至于这个问题对人有什么用，他可以暂时不管（虽然我也严重反对这种做法）。后来又有人来搞这个问题，多半是被老师分配来的。他也小修改一下，修改想法其实不费工夫，主要是你怎样把你的 Introduction 写好？可以让别人觉得你的工作有意义？这就是功夫。作家的功夫。我有一次面见 INRIA 的头目 Jean-Claude Paul 时，他就对我说："Tsinghua students are all writers, not scientists."

现在清华研究生做的事情无非就是，写好 paper，然后找个地方投出去。SCI 的最好，EI 的其次。偏僻的没人看的杂志也没关系，交钱也没关系。我就知道日本的一个 SCI 索引的期刊收 1000 美元的版面费。导师出钱，不投白不投，投了好毕业呵！

现在我也被"分配"来做这个问题。虽然说是一个有名的问题，但是这个有名的问题已经被研究了好几十年了。有很多挺厉害的人做出了很重要的贡献，但是我们为什么研究这个问题？我至今没有搞懂。

开头导师只是给了我两篇 paper, 据说是以前他一个得意门生写的，美国某大学的副教授。其中有一篇说是如何在不构造 Delaunay triangulation 的情况下生成 MST （最小生成树）。看到这篇文章开头说在 rectilinear metric 下，Delaunay triangulation 就不能用来构造 MST 了，所以他设计了一个新的算法。这个算法比起 Leo Guibas 的算法更加简单。文章里还提到一次 Matroid, 让初出茅庐的我觉得高深莫测。我还专门去借了一本《Matroid Theory》来看，其实他的论文剩下的部分跟 Matroid 没有任何关系。可是我对 "Delaunay triangulation 不能用来构造 RMST" 这个说法产生了怀疑。经过理论分析我觉得即使在 rectilinear metric 下，Delaunay triangulation 也可以用来构造 MST 的。我觉得作者只是故意这么写，想为他设计算法的动机找一个借口。我决定实践我的想法，写一个程序从 Delaunay triangulation 构造出一个 RMST。这本身不是什么创新的工作，可是我却在想，这样一个东西能不能用来构造 Steiner tree 呢？后来我真的就想出一个办法。实验表明我的算法比以前的算法要快几倍。

这是不是说我的算法是一个值得写 paper 的东西呢？导师说我应该写一篇，但是我认为我只是在挑别人的毛病时意外想出了一个改进的算法，并不会对将来的研究有什么启发。虽然程序快了一些，但是很少有那么大的线网需要这么快的算法，而且几倍的提高在我眼里不算是一个理论上的改进，而且这个算法不能推广到其他距离空间，可扩展性很低。所以我内心觉得这个结果不令我兴奋，不想写论文。但是在老师的一再要求下，我居然把这个研究写成了两篇 paper. 按照他的说法："应该分阶段总结你的成果。" 起初投出去的时候评委总是说这个东西不实用，导师说这是评委的问题，他们觉得不实用我们就投到理论一点的会议。经过几次投稿，还是失败了。我终于忍不住了，对副导师说出我的想法，我说："看一个作家的水平，是看他扔在垃圾筐里的纸。就让我把这篇 paper 永远藏在我的垃圾筐里吧。" 但是他不甘心，说你要相信自己的实力，然后把我的算法胡乱夸奖了一番。我说我不管了，随便你怎么办。我就开始研究我自己喜欢的东西去了。之后他居然真的投中一个欧洲的会议，是被 LNCS 收录的，LNCS 是 SCI 索引的，所以我居然有了一篇 SCI 文章！我自己不喜欢的文章也是 SCI 了！

第二篇论文就更传奇了。几投不中，就其原因，评委说是没有和现在“最先进”的算法程序实验比较。而我没有比较的程序，就是那个让我觉得发 paper 动机不纯的人的程序。没办法，求他给我代码。比了一下，确实比他快。不过我估计他程序写的有毛病，老是 core dump. 而且从实验数据来看，运行时间增长的速度不符合他论文里声称的时间复杂度。但是没办法，他只给 binary, 也不给源代码。程序快几倍，很有可能是实现上的问题，而不是算法更好。我的一个师兄以前就把他自己的算法戏称为“基于bug的优化”。我觉得这样比较对那个算法的作者不公平，完全没有发表的价值了。但是没办法，谁叫我们都是出来混的，没有人在乎这些。我还是记录下数据，添到论文上。一投就中，得了一个最佳论文奖。然后就有一篇校内新闻宣传：“我校王垠同学获得 XXX 会议最佳论文奖。这是大陆学者首次在如此高级别的会议上获得如此高的奖项。”这个“高级别”的会议，在我看来就是个垃圾。美国人都把最差的论文投到这里，就是为了来旅游一圈而已。

我对自己的做法产生了深深的负罪感，觉得自己正在进入这团混沌，正在被同化。我决定换一个题目研究。我就开始考虑 zero skew tree. 找了 20 多篇 paper 来看，发现他们没有什么本质的改进。而且对于问题本身的价值，他们完全就不清楚。有的作者后来甚至说，其实以前他们考虑的问题是没有必要解决的，因为实际应用中不可能遇到，我们其实可以把问题变成这样.....本来一句话就可以说清楚的事情，又写成了好几篇 paper. 我就是这样在 paper 的海洋中，找不到目标。

我见过的这种低级别的会议，低质量的论文几乎都是从 IEEE 那里出来的。道理很简单，IEEE 会议多，会议论文集都像两大块砖头，还是双列小字排版，当然能容纳下这么多的垃圾了。所以我对 IEEE 也没有好感。

1.14 火山小规模爆发

第一篇投中了会议之后，副导师很高兴的说“代替我去开会”，到希腊 Santorini 岛玩了一圈。回来还跟我说希腊不好玩，好苦啊，幸好你没去。然后就继续要我为算法申请一个专利。

写这个论文我都已经焦头烂额了，一点都不感兴趣。现在还要写专利，“要像教小学生做这件事一样，一步一步的把算法写清楚，举出实例”。我觉得快不行了，再这样折腾下去，我到博士毕业也许也就只搞出这些小儿科东西吧！我终于小规模爆发了一次。我坦荡的告诉了副导师我的想法，我觉得做学问应该是什么样，我觉得这么点东西不值得申请专利。我还告诉他我对国内的研究环境很失望。

他慌了，可能以为我想要退学，赶忙找我谈谈。对我说，我知道你心中有很大抱负。所以这次就不写专利了。我知道你想有更好的研究环境，但是不踏踏实实做好现在的工作，又怎么能有大的创造呢？然后就开始举爱因斯坦，居里夫人的例子..... 然后说，其实你在这里好好努力，将来出国的机会多的是，你想去 Harvard 也行，你想去 Princeton, 都行啊！

你说行就行？你去去给我看看？我们实验室从来就没有去这些地方的。继续这样做下去，以后哪个真正的科学家还会要我？

1.15 全面发展

在对清华的研究完全失望了之后。我就准备考 GRE, TOEFL 出国了。我去上了一个新东方的班，没学到什么英语方面的东西，倒是接触了很多新的思想。老罗的言论特别有趣，虽然我不是完全赞同他的意见。写 GRE 作文特别培养思维能力。我为了写 GRE 作文，常常为了一个不明白的问题到图书馆翻阅英文的哲学书籍，有关教育的书籍 ..... 对于很多问题我得到了完全不同的观点。大学的目的是什么？人的价值观是由理性决定的吗？等等等等。我读到了亚里士多德，柏拉图，康德等人的言论。甚至有个哲学家说 "All Animals Are Equal". 我看了他的文章觉得有很多可以批驳的观点。我看到迪卡尔的文章，说“要掌握科学就要掌握它的全部”，这句话真合我心意，我就是想做一个懂很多东西的人啊。我想结合艺术与科学。虽然我这个观点得到一些人的批判，但是我仍然相信迪卡尔。

从这些互相矛盾的观点中，我有了自己的判断力。我开始能够揭开从小蒙在我眼睛上的有色眼镜看问题。我开始检查我自己的思维，我以前的观点。看看它们是否是未经判断就盲目放进去的。我检查到很多很多的错误。我的待人接物，我对他人的理解上，都有不足之处。我还检查到妈妈传递给我的一些有色眼镜，小学课本给我们的有色眼镜。我开始学会用自己新的方式对待他人，看待事物。我不再盲目相信权威，哪怕他是诺贝尔奖得主，图灵奖得主。我有了自己的自由思维。

在那段时间，我感觉我的心智大门被开启了。我开始尝试从来没有做过的事情，以及从来不认为我能做好的事情。我一次又一次的相信我能。我能学会画画，我能打好太极拳，我能理解古典音乐..... 世界还有那么多美好的事情等着我去学习去开发啊！

可是，我们却像囚犯一样被判了5年在清华。博士学位就是我们的枷锁。

1.16 醒悟, paper 的奥秘

清华研究生谈论的重点是什么？是 paper. 吃饭的时候谈，喝茶的时候谈，睡觉的时候也谈。隔壁的同学在进校第一年就为 paper 惶惶不可终日，说：“你知道吗，他们要求我们发 SCI, 怎么办呢？我几个师兄都是因为没有 paper 延期毕业的。” 这恰好就是那个为后 10% 淘汰惶惶不可终日的同学。他的老师是个院士，可是他在手下就干一些写 word 文档之类的杂活还忙得要命，根本没有时间思考问题。

后来听说学校有规定，博士生必须发 4 篇 paper 才能毕业，其中必须有一篇是 SCI 索引，或者两篇 EI 索引。看上去冠冕堂皇的 SCI, EI, 不就是跟 google 差不多的东西吗？被它索引了怎么样了？特别是对文章的篇数作要求，而对质量没有判断。我其实读了两年都还不知道学校是这样规定毕业标准的。当我知道的时候，已经有人告诉我 SCI = Silly Chinese Index. 真是让人啼笑皆非。

学校没有能力评价学生的水平，就拿文章数来衡量。这样的毕业标准造就的是怎样的学生，怎样的实验室呢？难道导师真的没有能力判断 paper 的好坏吗？有些是，但是有些不是。即使他知道你的论文没什么价值，也会叫你发表。我发现 paper 数量的背后，是某些人的如意算盘。想一想是怎么回事吧。国家看什么来拨款研究？看 paper. 看什么来评价一个学校的水平，也是 paper. 国家没有能力评价你的能力，当然只有看你有多少 paper. 所以有了 paper 就有了钱。只要你能写 paper, 培不培养你，你将来的发展，关我们什么相干？你写的 paper 别人能不能看懂，能不能转化成生产力，管我们什么相干？怪不得有的院士想尽办法也要多收学生，宁愿自己帮学生出学费也要他进来。因为学生就是财源。paper 可以带来基金，可以在美国买小车洋房，没有基金就让学生干活吧。一个月几百块钱吊着一条命在那里为你拼命，谁叫他们想要那个博士学位呢！

该清醒了，博士无产阶级！

1.17 告别清华的博士学位

现在我已经厌烦了国内所谓的“学术”。我准备放弃清华的博士学位，出国找个好老师，进行真正的研究。博士第4年了，做出这样的决定真是不容易。有人告诉我不要放弃，你知道有多少人正在羡慕你？你知道一个清华的博士学位有多么值钱吗？但是我不能这么沉默下去了！

博士学位，累坏了多少年轻的中国人！我不再为它浪费我的青春。我知道国外大部分研究也不是那么好，如果国外也找不到好的老师，我就找一个简单的工作，和我心爱的人一起生活。有人说这是浪费人才？在清华混沌的过日子才是浪费呢！当一个侍者至少也让我感到对社会有贡献，看着顾客满意，我会露出笑容。可是做一个博士却没有。我感觉自己是个没用的人。

我已经完全看透了中国教育的失败。我高中的时候就受到它的伤害，这种伤害延续到现在。中国教育已经成为埋没人才的祸首。留在这个圈子里就是屈服，我不出声，大家都不出声，这个世界就会继续这样郁闷的运转下去。我今天要对这个系统大声地说一声“不！”

我离开了。可是中国永远也不缺少为清华拼命的人！因为他们的妈妈会告诉他们，清华是全中国最好的学校。你要考上清华，为我们光宗耀祖……

1.18 行动

2005 年 9 月 22 日下午 3 点，我在东主楼导师的办公室里跟导师和副导师再次重申了我的想法。包括以上的一切，和我准备退学，准备 10 月份考托福的打算。导师经过一番举例爱因斯坦，居里夫人，叫我踏踏实实的说教无效之后，严厉的批评了我只顾自己，不顾及教研组为我付出的心血。然后说：“要是你不能再为实验室作研究，我们就不能支持你了，前两个月实验室发的钱我收回。你可以马上写退学申请，我们实验室没有什么损失，我们有的是人干你的事情。不过我要告诉你，你一旦退学，连学校的住宿都要被收回！”

接着是副导师尖声的咆哮：“是啊，你瞧不起我们。我们是没有你聪明，可是我们勤勤恳恳……你知道你得的那个 best paper award, 我们付出了多少努力吗？你认为这么容易拿到吗？那是多少国外专家鉴定……”

我安静的等他说完。真像是一场闹剧，一场梦。他平息下来之后，我说了一声“再见”，然后默默地走出了办公室。

你们要退学申请？这里就是我的退学申请。

1.19 尾声

晚上收到副导师的 email 说：“还有一件事需要向你说一下：你在学校学习期间所取得的成绩包含你的努力、导师的指导帮助、同学们的帮助，还有学校和国家的支持。你作为博士生学习阶段取得的成果属于教研组、学校和国家。正如同我们作为职务发明的专利属于学校一样。

你在 MST, SMT 等方面取得了结果，它属于教研组、学校和国家。单位有责任进行合理的应用，为国家建设、国家荣誉服务。有责任进行进一步的整理丰富、向高水平的刊物投送。这里我们想说明一下上述的情况，同时，也告诉你一下：你若愿意将这些成果进行进一步的整理、我们已经给你提出了具体的修改意见，欢迎你按照进行修改。你若放弃，我们将进行具体的改进、投递。我们将尊重你的意见。谢谢。”

最后还是没有忘了 paper 的剩余价值。进一步验证了我的判断，他们在乎我吗？不。他们只在乎 paper. 至于我流离失所，又有何相干？我不知道有多少无知的弟弟妹妹又会把他们的研究建立在我不屑一顾的 paper 之上。

> Repair what you can — but when you must fail, fail noisily and as soon as possible. —Basics of The UNIX Philosophy

我不是一个中国教育操作系统下优良的程序。我在系统里运行了将近 20 年，快到最后的时候才喧闹的退出, dump 出这么大一个 core file. 我知道有的程序很早就退出了，我自愧不如他们。但是有的程序一声不响就退出了，还有很多的程序成为了 zombie, 永远的驻留在系统中成了系统的负担，在这一点上我又比他们好一些。至少我让程序员有机会用调试器检查 core 文件，调查这个程序运行中哪里出了问题。

“你退学就退学，干吗大惊小怪，牢骚满腹的？”如果只是有牢骚，我就把隔壁同学拉过来一起发发牢骚就完事了。可是我虽然不是优秀的程序，我觉得应该为修复这个系统，修复自己做点什么。我希望国家的教育和研究环境好起来，这样大家就安心的生活，不用出国搞得奔波流离。有多少恋人由于一个人出国了而痛苦的分手，有多少父母在盼望海外游子的归来？我不能像很多人那样申请了国外的学校，拍拍屁股就走人。我一年前就考 GRE 想出国，可是我总是自欺欺人的幻想国内的境况会好起来，有时我觉得看到希望，可是马上希望又破灭了。一个个大师来了，让我一次次燃起希望，可是发现他们对环境的作用也不大。一些大师不满意，又走了。我自己也想尽力改造环境，结果经过多次努力无效，自认能力不够，终于放弃了。

在发现大家都忙着发表 paper 而没有讨论时，我曾经建议设立一个清华的 THU-Technical Report. 我的想法是：最差的草稿扔在垃圾堆里；可能有用但是还不值得向所有人公开的东西发到 THU-TR, 供系内查阅；如果发现 THU-TR 的东西会有用，再好好修改了转投会议或者期刊。系学术助理王磊很高兴的采纳了我的建议，并且自愿维护一个 THU-TR 的编号。可是根本没有人愿意把自己好不容易写出来的但是确实又不值得发表的东西投到这里，因为世界上总有地方可以把这个东西投出去，还是 SCI 和 EI, 而这个 THU-TR 连正式刊物都不算。后来有人告诉我，如果学生都把东西投到我们这里，不知道有多少导师会跟我们急。所以 THU-TR 的计划就这么告罢。

我一年前写信给 Knuth，这个我相信是真正的大师。我说我想退学，想请他推荐一些真正的研究者给我做老师。他回信说“你先找精通中国文化的长者谈谈”。我意识到他可能觉得这是一个文化的问题。我于是想知道中国的科技为什么搞不好，就开始看一些有关文化的东西。后来居然跑到中国社会科学院去听新竹清华大学人文学院的院长讲座，后来又在清华参加了人文学院的研讨会。会上一个老师说的好，当一个制度没法衡量学术水平本身，它就会用一个似乎等价的标准，比如 paper 数或者高考分数。但是一旦这个标准被确立，人们就会向着这个标准努力，而不是向学术水平本身。他们总会发现制度的很多问题，找出破绽，去达到这个标准，而不是提高自己的学术水平。最后，这个标准已经完全不能反映水平本身。我就在想，这个问题大了，这不仅是环境，制度，而且还是长久以来的文化造成的。从新竹清华大学院长的讲座里，我发现英国人是怎样用科学技术打开了中国的大门，而乾隆皇帝是如何对科学不感兴趣。中国似乎从古到今就不重视科学技术的，中国有自己的优势，自己的文化。对啊，科学技术是个双刃剑，如果照美国那样发展下去也不知道会怎么样。我们中国的文化是瑰宝，但是它已经被外国的坚船大炮打得遍体鳞伤。这不是我们的错，但是我们要努力恢复自己的文化，不能总是怨天尤人。我就开始看道德经之类的东西，还去西麓学社参加古代文化讨论活动，后来又开始打太极拳。

我觉得再没有从实际出发的目标，我的研究就会完全变成纸张了，就像我高中感觉到的一样。所以后来我就自己设立了一个研究方向，我把自己称为“研究博士生”，我要去了解博士生都是怎么样生活的。我就想知道有多少学生有跟我类似的困境。我跟很多朋友谈过，去了解他们的苦衷，研究生也有，本科的也有。我觉得我还应该了解更多的人，就试图到研究生通讯社做记者，心想挂一个记者证，就好跟人套磁问一些问题了。结果他们说我口才不好，所以做了一个秘书。后来记者们告诉我，他们是由上级分配任务的，根本不可能让你去报道学生真正的想法。我为了多多接触外国文化，比较中西文化的不同，又加入了学生对外交流协会(ASIC)，我在 ASIC 有了很多好朋友。博士生论坛的时候也有很多同学跟我反映研究上的问题。讨论成立特别兴趣小组（SIGs）的时候，我就提议成立一个 Common Room, 一个同学说她去 Stanford 的时候那里就有很好的 Common Room, 很多人在一起讨论，这是国外大学斯通见惯的东西。我告诉 Oxford 的朋友我的想法，他很惊奇地说：“你们居然没有 Common Room？” 后来吃饭时我又找一些老师谈话，发现他们也对这个事情无可奈何。老师自己的办公室都要自己出钱，谁还能支持你们有这么大一个房间？而且即使有了房间，谁来讨论？还不就是拿着别人的 paper, 试图找点可以改进的地方，或者就讨论哪个会议好发 paper. Common Room只是一个形式，只要有人感兴趣，随便找个茶馆也能讨论。问题就在于没有人有精力有心情进行真正的讨论，制度决定一切。我们无能为力。我觉得自己一个学生力量太小，曾经试图找大师帮忙。我找到 Andy Yao, 述说我的苦衷。结果他对我说：“别试图去改造环境！你没有这个能力，连我都没有！改造好你自己就不错了。" 改造好我自己，可是怎么改？所以我决定先换一个环境，到一个真正搞研究的地方去体会，去学习。

其实我不后悔进入川大，不后悔来到清华，珍惜一切的历史，因为没有它们，我也许就不是现在的我，有着自己想法的我。我也许就在安逸的生活中变得堕落。它们不完美甚至给我痛苦，但是我还是珍惜，珍惜这里的朋友，这里的一草一木。也许这就叫做爱。我会变得更好，我会挂念我的满目苍夷的祖国母亲。我会回来告诉你我学到的一切，我会给你和其他儿女真正的幸福，一定的！

2 我和权威的故事

每个人小时候心里都是没有权威的，就像每个人小时候也都不相信广告一样。可是权威就像广告，它埋伏在你的潜意识里。听一遍不信，听两遍不信，.....，直到一千遍的时候，它忽然开始起作用了，而且这作用越来越强。

消灭广告所造成的幻觉，最好的办法就是去尝试，去实地的考察它。有些虚幻的东西只要你第一次尝试就会像肥皂泡一样破灭掉。可是如果你不主动去接触它，它就会一直在你脑海里造成一种美好神圣的假象。越是得不到的越是觉得美好。很神奇的一个现象就是，权威对人思想的作用其实也跟广告一样。

上大学以前的人因为没有专业，所以还不怎么崇拜权威，大不了追追歌星，影星，球星啥的。而进了大学之后，就会开始对本领域的权威耳濡目染。一遍，两遍，一千遍的听到同学们仰慕某“牛人”或者“大师”的名字，虽然从来没亲身见过，不知不觉就对这人产生了崇拜心理，然后自愧不如。不知不觉的，自己也开始附和这些说法，不自觉地提到这些大师的名字，引用他们说的话作为自己的行动指南。

Donald Knuth, Dennis Ritchie, Ken Thompson, Rob Pike, ... 就是通过这些途径成为了很多计算机学生的权威。以至于几十年以后，他们的一些历史遗留下来的糟糕设计和错误思想还被很多人奉为神圣。

2.1 Donald Knuth

很多人（包括我）都曾经对 Knuth 和他的 The Art of Computer Programming (TAOCP) 极度崇拜。在我大学和研究生的时候，有些同学花了不少钱买回精装的 TAOCP 全三卷，说是大概不会看，但要供在书架上，镇场子。当时我本着“书非借不能读也”的原则，再加上搬家的时候书是最费力气的东西，所以坚决不买书。我就从图书馆把 TAOCP 借了来。说实话我哪里看的下去啊？那里面的程序都是用一个叫 MIX 的处理器的汇编语言写的。一个字节只有 6 位，每位里面可以放一个十进制数（不是二进制）！还没开始写程序呢，就开始讲数学，然后就是几十页的公式推导，证明..... 接着我就睡着了。但我总是听说有人真的看完过 TAOCP, 然后就成为了大师。比尔盖茨也宣称：“要是谁看完了 TAOCP, 请把简历投给我！”在这一系列的号召和鼓吹之下，我好几次的把 TAOCP 借回来，下定决心这次一定看完这旷世奇书。每次都是雄心勃勃的开始，可从来就没看完过开头那段 MIX 机器语言和数学公式。

看不懂 TAOCP 总是感觉很失败，因为看不懂 TAOCP 就成不了“大师”，可我仍然认为 Knuth 就是计算机科学的神，总能从他那学点什么吧，所以又开始折腾他的其他作品。这就是为什么我开始用 TeX, 并且成为中国 TeX 界的主要“传教士”之一。为了 TeX, 我把 Knuth 的 TeXbook 借回来，从头到尾看了两遍，做完所有的习题，包括最难最刁钻的那种 “double bend” 习题。接着又开始看 MetaFont Book, 开始使用 MetaPost 进行绘图。开头还挺有成就感，可是不多久就发现学会的那些 TeX 技巧到了临场的时候就不知道该怎么用，然后就全都忘记了。这就是为什么我把 TeXbook 看了两遍，可是看完第二遍之后不久还是忘记得一干二净。

师兄师姐看到我用 TeX，说怎么折腾这么过时的玩意儿。我很气愤他们以及国内学术界居然都用 Word 排版论文，就开始针锋相对，写出一系列煽动文章鼓吹 TeX 的种种好处，打击“所见即所得”这种低智商玩意儿。这还不够，又开始折腾 Knuth 设计的 MMIX 处理器，并且认为 MMIX 的寄存器环就是世界上最先进的设计。发现一些无关紧要的小错，就给 Knuth 发 email, 居然拿到两张传说中的 “Knuth 支票”, 并且一度引以为豪。当然像所有拿到 Knuth 支票的人一样，你是不会去兑现它的，甚至有人把它们像奖状一样放在相框里。我还没那么疯狂，那两张支票一直在它们原来的信封里。多年以后我到美国想兑现那支票的时候，发现它们已经过期了。

图 1.

当你心里有了这样的权威，其他人的话你是不可能听得进去的，就算他们其实比你心目中的权威更具智慧也一样。在清华的时候我有时候去姚期智的小组听客串讲座。有一次请来了美国某大学一个教授讲算法，不知道怎么的我就跟他聊起 TAOCP, 大概是想请教他如何学习算法。他跟我说 Knuth 的书已经比较过时了，你可以看看 MIT 的那本《算法导论》。可是这位教授的名气怎能和 Knuth 相比，这话我恁是没有听进去，仍然认为 TAOCP 隐藏了算法界最高的机要，永恒的珍宝。

在清华的时候我很喜欢一门叫做"计算几何"的课，就经常跟那门课的老师交流思想。有一次我在 email 里面提到 Donald Knuth 是我的偶像，那位老师很委婉的回复道："有偶像很好啊，Knuth 也曾经是我的偶像。" 我对"曾经"这两个字感到惊讶：难道这意味着 Knuth 现在不是他的偶像了？在我执意的询问之下他才告诉我，其实世界上还有很多更聪明的人，Knuth 并不是计算机科学的一切。你应该多看看其他人的作品，特别是一些数学家的。然后他给了我几个他觉得不错的人的名字。

现在回想起来，这些话对我是有深远作用的。那位老师虽然在系里的"牛人"们眼里是个研究能力（也就是发 paper 能力）不强的人，但是他却对我的人生转折有着强有力的作用。他引导了我去追寻自己真正的兴趣，而不是去追寻虚无的名气。我发现很多人都在为着名气而进行一些自己其实不感兴趣的事情，去做一些别人觉得"牛气"的事情。我真希望他们遇到跟我一样的好老师。

在现在看来，Knuth 的 TAOCP 就是所谓的 "神圣的白象" (white elephant). 大家都把它供起来，其实很少有人真的看过，却要显得好像看过一样，并且看得津津有味。这就让试图看懂它的人更加自卑和着急，甚至觉得自己智商有问题。别人都看过了，我怎么就看不懂呢？其实 TAOCP 里面的大部分算法都不是 Knuth 自己设计的，而且他对别人算法的解释经常把简单的问题搞得很复杂。再加上他执意要用汇编语言，又让程序的理解难度加倍。

有一句话说得好："跟真正的大师学习，而不是跟他们的徒弟。"如果你真的要学一个算法，就应该直接去读那算法的发明者的论文，而不是转述过来的"二手知识"。二手的知识往往把发明者原来的动机和思路都给去掉了，只留下苍白无味，没有什么启发意义的"最后结果"。确实是这样的，多年以后当我看见 Knuth 计划中的几卷新的 TAOCP 的目录时，发现其中大部分的东西我已经通过更容易的方式学到了，因为我找到了这些知识的源头。

所以之前的那位访问清华的教授说的其实是实话，Knuth 真的落伍了，可是就算在美国也少有人知道或者承认这个情况。有一次看一个对世界上公认最厉害的一些程序员的采访，包括总所周知的一些大牛，以及 ML 的设计者 Robin Milner, Haskell 的设计者之一 Simon Peyton Jones 等人。也不知道采访者是什么心理，在对每个人的采访中他都问，你看过 TAOCP 吗？大部分人都说看过，真是了不起的巨著，很重要啊云云。只有 Robin Milner （如果我没记错的话）比较搞笑，他说我希望我看过，但是可惜实在没时间。我一直把 TAOCP 垫在我的显示器下面，这样我工作时就可以一直看着它们 :)

Knuth 说 "premature optimization is the root of all evil", 然而他自己却是非常喜欢用 premature optimization 的人。他的代码里到处是莫名其妙的小聪明，小技巧。把代码弄得难懂，实际上却并没有得到很多性能的提高。有一次看 MMIX 处理器的模拟程序，发现他用来计算一个寄存器里的 "1" 的个数的代码非常奇怪。本来写个循环，或者用那种从末位减 1 的做法就可以了，结果他的代码用了 Programming Pearls 里面一个古怪的技巧，费了我半天时间才看懂，后来我发现这个技巧其实还不如最简单的方法。就是这些细小却又蹊跷的做法，使得 Knuth 的代码用细节掩盖了全局，所以到最后我其实也没从大体上搞懂一个处理器的模拟器应该如何工作。直到后来到 Indiana 学习了程序语言的理论之后我才发现，其实处理器模拟器（以至于处理器本身）的工作原理很简单，因为它就是一个机器代码的解释器。使用跟高级语言解释器同样的结构，你可以比较容易的写出像 MMIX 模拟器那样的东西。

Knuth 最重要的一个贡献恐怕是程序语言的 parsing (语法分析), 比如 LR parsing, 然而 parsing 其实是一个基本不存在的人造问题。它的存在是因为人们的误解，以为程序语言需要有跟人类语言一样的语法，所以把程序语言搞得无端的复杂和困难。如果你把语法简化一下，其实根本用不着什么 LR，LALR. 我最近给我自己设计的语言写了一个 parser , 从头到尾只花了两个小时，500 行 Java 代码，包括了从 lexer 一直到 AST 数据结构的一切。完全手写的代码，根本没用任何复杂的 parsing 技术和 YACC 之类的工具，甚至正则表达式都没有用。之所以可以这样，因为我的语法设计让 parsing 极其容易，比 Lisp 还要容易。Knuth 过度的强调了 parsing. 他的误导使得很多人花了几十年时间来研究 parser, 到现在还在不时地提出新的技术，用于设计更加复杂的语法。何必呢？这只会让程序员和编译器都更加痛苦。如果这些人把时间都花在真正的问题上，那今天的计算机科学不知道要美好多少。

几乎每一本编译器教材都花大量篇幅来讲述 DFA, NFA, lexing, LL, LR, LALR..... 几乎每个学校的编译器课程都会花至少 30% 的时间来做 parser, 折腾 LEX，YACC 等工具，而对于编译器真正重要的东西却没有得到很多的训练。这就是为什么 Kent Dybvig 的编译器课程如此有效，因为 Scheme 的语法非常简单，我们根本没有花时间来做 parser。我们的时间用在了思考真正的问题：做优化，实现尾递归，高阶函数..... 很多语言梦寐以求却又做不好的东西。这样的课程给了我可以发挥自己潜力的余地，我的课程编译器里面具有大量的独创写法，我的 X64 机器代码生成器生成极其短小的代码，让 Kent Dybvig 都在背地里琢磨是怎么回事。这些东西到现在也许仍然是世界上最先进的技术。

一个人的思维方式似乎决定了他设计的所有东西。Knuth 的另一个最重要的发明，文学编程（Literate Programming）其实也是多此一举，制造麻烦。文学编程的错误在于认为程序语言应该像人类语言，应该适应所谓的“人类思维”。然而程序语言却是在很多方面高于人类语言的，它不应该受到人类语言里的糟粕的影响。把程序按照 Knuth 的方式分开在不同的文章段落里，造成了代码之间的关系很难搞清楚，而且极其容易出错。这个错误与 “Unix 哲学” 的错误类似，把程序作为一行一行的文本，而不是一个像电路图一样的数据结构。我不想在这里细说这个问题，对此我专门写了一篇文章，讲述为什么文学编程不是一个好主意。

TeX 其实也是异常糟糕的设计。它过度的复杂，很少有人搞得懂怎么配置。经常为了一个简单的效果折腾很久，然后不久就忘了当时怎么做的，回头来又得重新折腾。原因就是因为 TeX 的设计缺乏一致性，特殊情况太多，而且组合（compose）能力很差。所以你需要学太多东西，而不是跟象棋一样只需要学习几个非常简单的规则，然后把它们组合起来形成无穷的变化。

在程序语言设计者看来，TeX 的语言是世界上最恶劣的设计之一，但如果没有这个语言，它也许会更加糟糕。其实 TeX 之所以有一个“扩展语言”，有一个鲜为人知的小故事。在最早的时候 Knuth 的 TeX 设计里并没有一个语言。它之所以有一个语言是因为 Scheme 的发明者 Guy Steele. Knuth 设计 TeX 的那个时候 Steele 碰巧在斯坦福实习。他听说 Knuth 要设计一个排版系统，就建议他设计一个语言，以应付以后的扩展问题。在 Steele 的强烈建议和游说之下，Knuth 采纳了这个建议。可惜的是 Steele 并没能直接参加语言的设计，在短短的一个夏天之后就离开了斯坦福。

Knuth 的作品里面有他的贡献和价值，TeX 的排版算法（而不是语言）也许仍然是不错的东西。可是如果因为这些好东西爱屋及乌，而把他所推崇的那些乱七八糟的设计当成神圣的话，那你自己的设计就逃脱不出同样的思维模式，让简单的事情变得复杂。仍然对 TeX 顶礼膜拜的人应该看一下 TeXmacs，看看它的作者是如何默默无闻的，彻彻底底的超越了 TeX 和 Knuth.

在我看来，Knuth 是个典型的精英主义者，他觉得自己做的都是最好，最有“格调”的。他利用自己的权威和特立独行来让用户屈服于自己繁复的设计，而不是想法设计出更加易用的工具。TeX 的版本号每次更新都趋近于圆周率 π, 意思是完美，没有 bug. 他

奖励大额的支票给发现 TeX 代码里 bug 的人，用于显示自己对这些代码的自信，然而他却“冰封”了 TeX 的代码，不再填加任何新东西进去，也不再简化它的设计。当然了，如果不改进代码，自然就不会出现新的 bug，然而它的设计也就因此固步自封，停留在了几十年以前。更奇怪的是，"TeX" 这个词居然不按照正常的英语发音逻辑读成 "teks". 每当有人把它“读错”，就有“高手”打心眼里认为你是菜鸟，然后纠正：“那个词不读 teks, 而要读‘特喝’，就像希腊语里的 chi, 又像是苏格兰语的 loch, 德语的 ach, 西班牙语的 j 和俄语的 kh.” 也许这就叫做附庸风雅吧，我是纯种的欧洲人！;-) 当一个软件连名字的发音都这么别扭，这么难掌握，那这个软件用起来会怎样？每当你提到 TeX 太不直观，就有人跟你说："TeX 是所想即所得，比你的所见即所得好多了！" 可事实是这样吗？看看 TeXmacs 吧，理解一下什么是“所见即所得+所想即所得”二位一体。

我跟 Knuth 的最后一次“联系”是在我就要离开清华的时候。我从 email 告诉他我觉得中国的研究环境太浮躁了，不是做学问的好地方，想求点建议。结果他回纸信说：“可我为什么看到中国学者做出那么多杰出的研究？计算机科学不是每个人都可以做的。如果你试了这么久还不行，那说明你注定不是干这行的料。”还好，我从来没有相信他的这段话，我下定了决心要证明这是错的。多年的努力还真没有白费，今天我可以放心的说，Knuth 你错了，因为我已经在你引以为豪的多个方面超过了你。

2.2 Unix

Unix 的创造者们是跟 Knuth 非常类似的权威，他们在我的心目中也曾经占据了重要的位置，以至于十年前我写了一篇文章叫《完全用 Linux 工作》，大力鼓吹 Unix 的“哲学”，甚至指出 Linux 不能做的事情就是不需要做的，并且介绍了一堆难用的 Unix 工具，引得很多人去折腾。可如果你知道我现在对 Unix 的态度，肯定会大吃一惊，因为在经过努力之后，我成功的“忘记”了 Unix 的几乎一切，以至于本科刚毕业的学生都会以为我是脑盲，并且以为可以在我面前炫耀自己知道的 Linux 技巧。他们不会明白，在我心里 Unix/Linux 的设计是计算机软件界目前面临的大部分问题的罪魁祸首，而他们显示给我看的，只不过是 Unix 的思想和精英主义给程序员造成的精神枷锁。其实我并不会忘记 Linux 的设计，但我已经下意识的以熟悉 Linux 的奇技淫巧为耻，所以很多时候我即使知道也要装作不知道。因为我是机器的主宰，而不是它的奴隶，所以我总是想办法让机器去帮我做更多的事，帮我记住那些无聊的细节，而不是去顺从它的设计者所谓的“哲学”。

评论 Unix 和它的后裔们总是一件尴尬的事情，因为你提到它们的任何一个缺点，都会被很多人认为是优点。GNU 的含义是 “GNU is Not Unix”, 但很可惜的是 GNU 和 Linux 的设计从来没有摆脱过 Unix 思想的束缚。Unix 的内存管理，进程，线程，shell，进程间通信，文件系统，数据库..... 几乎都是很蹩脚的设计。所谓的 "Unix 哲学 ", 也就是进程间通信主要依靠无结构字符串，造成了一大批过度复杂，毛病众多的工具和语言的产生：awk, sed, perl,..... Unix 的内存管理是按“页”而不是按“结构”分配，相当于把内存分配的任务完全推给应用程序。而且允许任意的指针操作，这就像给每个老百姓一把爱走火的枪。可是又想要“安全”，自相矛盾。没办法，不得不强制进程数据空间完全隔离，使得进程间无法直接传递数据结构。进程和线程上下文切换开销过大，造成了使用大规模并发或者分布式计算的瓶颈，导致了 goroutine 和 node.js 等“变通方法”的产生。把数据无结构的存储在文件里，无法有效的查找数据，造成了关系式数据库等过度复杂的数据解决方案的产生。再加上后来 WEB 的设计，现在的网站基本上就是补丁加补丁，一堆堆的 hack.

"Unix 哲学" 貌似也有好的部分，比如“每个程序只做一件事，多个程序互相合作。”然而，这个所谓的哲学其实就是程序语言（比如 Lisp）里面的模块化设计。它当然是好东西，然而这些思想被 Unix 偷来之后，有其名而无其实。很少有 Unix 程序真正只做一件事的，而且由于字符串这种通信机制的不可靠，它们之间其实不能有效地合作。有时候你换了一个版本的 make 或者 sed 之类的工具，你的 build 就莫名其妙的出问题。这就是为什么有的公司请了专门的所谓 “build engineer”, 因为高级别的程序员不想为这些事情操心。Lisp 程序员早就明白这个道理，所以他们尽一切可能避免使用字符串。他们设计了 S 表达式，用于结构化的传输数据。实际上 S 表达式不是“设计”出来的，它是每个人都应该首先想到的，最简单的可以表示树结构的编码方法。Lisp 的设计原则里面有一条就是：Do not encode. 它的意思是，尽量不要把有用的数据编码放进字符串。Unix 的世界折腾来折腾去，XML,CORBA,..... 最后才搞出个 JSON, 然而其实 JSON 完全不如 S 表达式简单和强大。Unix 就像一个脑瘤，它让人们放着最好的解决方案几十年不用，不断地设计乌七八糟的东西用来取代乌七八糟的东西。这些垃圾对人有很大的洗脑作用。前段时间我说 S 表达式比 JSON 简单，有人居然跟我说 JSON 好些，因为它结构的 field 是“无顺序”的。这让我相当无语，因为一个编码方式有没有顺序完全取决于你如何解释它。从这个意义来讲，S 表达式可以是有顺序，也可以是没有顺序的。

Unix 喜欢打着“自由”和“开源”的旗号，可是它的历史却充满了政治，宗教，利益冲突和对“历史教科书”的串改。几乎所有操作系统课本的前言都会提到 Unix 的前身 Multics, 而提到 Multics 的目的，都是为了衬托 Unix 的“简单”和伟大，接下去基本上就是按部就班的讲 Unix 的设计，仿佛 Unix 就是世界上唯一的操作系统一样。课本会告诉你，Multics 由于设计太复杂，试图包罗万象，最后败在了 Unix 手下。可是如果你仔细了解一下 Multics 的历史，就会发现最后一台 Multics 机器直到 2000 年还在运行，拥有 Unix/Linux 到现在还没有的先进而友好的特性，并且被它的用户所爱戴。Multics 的设计并不是没有问题（对比一下 Lisp Machine 和 Oberon），但是相比之下，Unix 的设计一点都不简单。Unix 抄了 Multics 最好的一些思想，有些没有抄得像，然后又引入了很多自以为聪明的糟粕。可是 Unix 靠着自己病毒一样的特征，迅速占领了市场。Unix 最开头是开源和免费的，但是后来 AT&T 发现这里面有利可图，所以就收回了使用权，并且开始跟很多人打官司。AT&T 的邪恶比起微软来，真是有过之而无不及。

Unix 的很多设计是如此龌龊，很多人却又由于官僚的原因不得不用它。以至于 Unix 出现的早期怨声载道，有人甚至组织了一个 mailing list 叫 "Unix 痛恨者" (Unix Haters). 你很有可能把这些人当成菜鸟，可是这些人其实都用过更好的操作系统，有的甚至设计实现过更好的操作系统甚至程序语言。最后他们的叫骂声被整理为一本书，叫做 Unix Hater's Handbook. 让人惊讶的是，这本书有一个“反序言” (anti-foreward), 作者正是 Unix 和 C 语言的设计者之一，Dennis Ritchie. 这个反序言说，Unix 这座设计缺乏一致性的监狱会继续囚禁你们，聪明的囚犯会从它里面找到破绽，可惜的是自由软件基金会（FSF）会建造跟它完全兼容的监狱，只不过功能多一些。拥有三个 MIT 学位的记者，微软的研究员，Apple 的高级科学家可能还会对这座监狱的“规矩”贡献一些文字。从这些文字里，我看到了一个炫耀武力的暴君，看到了赤裸裸的权威主义和教条主义。

可惜的是在软件的世界里任何糟糕的设计都可以流行，只要你的广告做得好，只要你的传教士够多。一知半解的人（比如十年前的我）最喜欢到处寻找“新奇”的东西，然后开始吹嘘它们的种种好处，进而成为它们的布道者。再加上大学计算机系的“紧跟市场”的传统，不幸的事情发生了：Unix 和它的后裔们几乎垄断了服务器操作系统的市场。由于 Unix 的垄断，现在的软件世界基本上建立在一堆堆的变通之上，并且固化之后成为了“珍珠”。公司里，学校里，充满了因为知道一些 Unix 的“巧妙用法”而引以为豪的人，殊不知他们知道的只是回避一些蹩脚设计的小计俩。程序员有太多的特例和细节需要记

忆，不但不抱怨，还引以为豪。很少有人想过如何从根本上解决问题，历史的教训很少有人吸取，以至于几十年前犯过的设计错误还在重现。Unix 的最大贡献，恐怕就是制造了大量的工作岗位—因为问题太多太麻烦，所以需要大量的人力来维护它的运行。

现在看来，Unix 当初就是依靠《皇帝的新装》里织布工的办法封住了大家的嘴。皇帝的织布工们说：“愚蠢或者不称职的人都看不见这件衣服。”Dennis Ritchie 说："Unix 是简单的，但只有天才才能理解这种简单." 看出来了吗？你不敢说 Unix 的设计太乱太复杂，因为这话一出口，立马会有人引用 Dennis 的话说，是你自己不够天才，所以不理解。当然了，这就意味着他比你聪明，因为只有天才才能理解这种简单嘛。哎，这种喜欢显示自己会用某种难用工具的人实在太多了。你不敢批评这些工具对用户不友好，因为你立即会被鄙视为菜鸟。

Dennis Ritchie 去世了。死者长已矣，可是有些他的崇拜者在那个时候还要煽风点火，拿他的死与 Steve Jobs 的死来做对比，把像这样的照片四处转帖，好像 Steve 死错了时间，抢了 Dennis 的风头似的。然后就有人写一些这样的文章，把世界上的所有系统，所有语言都归功到 Dennis 和 Unix 身上。看到这些我明白了，所谓的“天才”就是这样被造出来的。在我看来这些是很滑稽的谬论，就像是在说有人拿一把很钝的剪刀做出了一件精美的衣服，所以这剪刀立下了汗马功劳。其实这人一边裁布一边在骂这剪刀，心想妈的这么难用，快点做出这衣服，卖了钱买把好点的！

用了这么久 Apple 的产品，平心而论，虽然它们并不完美，然而它们并不是 Unix 的翻版，它们做出了摆脱 Unix 思想束缚的努力。它们本着机器为人服务的原则，而不是把人作为机器的奴隶。Mac 的很多内部设计跟 Unix 有着本质的不同。然而就是这样的系统，被 Dennis Ritchie 在他的反序言里面蔑称为“以 Sonic the Hedgehog 作为智力主题和交互设计基础的系统”。

有谁知道，在那同样一段时间里，Lisp 的发明者 John McCarthy, ML 的发明者 Robin Milner, 都相继去世了呢？那个时候我只是在 mailing list 看到有人发来简短的消息，然后默默地思念他们给我带来的启迪。我们没有觉得 Steve Jobs 的死抢了他们的风头，因为他们不需要风头。死就是要安安静静的，让知己者默哀已经足矣。出现这种事情恐怕不能怪 Dennis Ritchie 自己，然而这些 Unix 的崇拜者们，真的应该反省一下自己的做法了。

Unix 的设计者们曾经在我的心里占据了一席之地，可是现在觉得他们其实代表了反动的力量，他们利用自己的影响力让这些糟糕的设计继续流传，利用人们的虚荣心，封住大部分人的嘴，形成教条主义，让你认为 Unix 的设计是必须学习的东西。很多人成为了 Unix 的传教士和跟屁虫，没有什么真实水平，就会跟着瞎起哄，把 Unix 设计者的话当成教条写进书里。可是他们的权威和名气是如此之大，让我在很多人面前只能无语。

2.3 Go 语言

现在，同样这帮 Unix “大牛”们设计了 Go 语言，并且依仗自己的权威和 Google 的名气大力推广。同样的这帮跟屁虫开始使用它，吹捧它，那气势就像以为 Go 可以一统天下的样子。真正的程序语言专家们都知道，Go 的设计者其实连语言设计的门都没摸到。这不是专家们高傲，他们绝不会鄙视和嘲笑一个孩子经过自己的努力做出一个丑陋的小板凳。他们鄙视，他们嘲笑，因为做出这丑陋小板凳的不是一个天真的小孩，而是一些目空一切的人，依仗着一个目空一切的公司。他们高举着广告牌，试图让全人类都坐这样丑陋的板凳。

跟当年设计 Unix 时一个德行，不虚心向其它语言和系统学习经验教训，就知道瞎猜瞎撞。自己想个什么就是什么，但其实根本就不知道自己在干什么。把很多语言都有的无关紧要的功能（比如自动格式化代码）都吹嘘成是重大的发明，真正重要的东西却被忽略。Go 语言的设计在很多方面都是历史的倒退，甚至犯下几乎所有其他语言都没有的低级错误。在语法上大做花样，却又搞得异常丑陋，连 C 和 Java 都不如。自己不理解或者实现难度大点的东西就说是不需要的，所以连很多语言支持的 parametric type（类似 Java generics）都没有，以至于没法让程序员自定义通用数据结构，只好搞出一堆特例（比如 map，make，range）来让程序员去记。这些做法都跟 Unix 如出一辙。

Go 语言最鲜明的特征就是 goroutine，然而这个东西其实每个程序语言专家都知道是什么。有些语言比如 Scheme 和 ML 提供了 first-class continuation（call/cc），可以让你很容易实现像 goroutine 这样的东西，甚至实现硬件中断的"超轻量线程"。至于 Go 那种"基于接口"的类型系统设计，我在很多年前就已经试验过，并且寄予了很大的希望。结果最后经过很多的研究和思索后发现有问题，于是放弃了这个想法。很显然，我不是第一个在这个问题上失败的人，很多语言专家在使用 parametric type 以前都试图过做这种基于接口的设计，结果最后发现不是什么好东西，放弃了。然而 Go 的设计者却没有学到这些失败教训，反而把它当成宝贝。一个很显然的问题是，在 Go 里面你经常会需要使用"空接口"（interface{}），用来表示所有类型。这就像使用 C 的 void 指针一样，有着静态类型系统的麻烦，却失去了静态类型系统的好处。

每当你提到 Go 没有 parametric type，Go 的拥护者们就说"我看不到这有什么用处"，就像一些非洲土著跟你说"我看不到鞋子有什么用处"一样。他们利用人们对 Java 的繁复和设计模式的仇恨，让你抛弃了它里面的少数好东西。其实 Java generics 不是 Java 首先有的。它的主要设计者其实包括 Haskell 的设计者之一 Philip Wadler. 这种 parametric type 很早就出现在 ML，Haskell 等语言里面，是非常有用的东西。

每当受到批评，Go 的拥护者们就托词说，Go 是 "系统语言" (systems language). 这里潜在的前提就是，认为 Unix 就是唯一的"系统"，而 C 就是在 Go 以前唯一的"系统语言"，好像其他语言就写不出所谓的"系统"似的。而事实是，在 C 诞生十年以前，人们就已经在用 Algol 60 这样的高级语言来写操作系统了。由于先天不足却又大力推广，所以 Go 的很多缺陷基本已经没法修补了。这样的语言一旦流行起来就会像 Unix 一样，成为一个无休止的补丁堆。如果像 Java 或者 Haskell 这样的语言还值得批评的话，对 Go 语言的设计者我只能说，去补补课吧。

2.4 Cornell

可是权威和名气的威力还是很大的。虽然 Knuth 在我心目中的位置不再处于"垄断地位"，世界上可以占据我心里那个位置的人和事物还很多。在离开清华之后我申请了美国的大学。也许是天意也许是巧合，只有两所大学给了我 offer：Cornell 和 Indiana，而我竟然先后到了这两所大学就读。

说实话，Indiana 给了我比 Cornell 更好的 offer. Cornell 给我的是一个 TA 的半工读职位，而 Indiana 给我的是一个不需要工作白拿钱的 fellowship. 说实话我从来没有搞明白 Cornell 这样的"牛校"怎么会给我这样的人 offer, GPA 一般，paper 很菜，而 Indiana 却是真正在乎我的。Indiana 的 fellowship 来自 GEB 的作者 Doug Hofstadter. 他从 email 了解到我的处境和我渴求真知的愿望之后，毅然决定给我，一个素不相识的人写推荐信。后来我才发现那 fellowship 的资金也是他提供的。

可是 Indiana 和 Hofstadter 的名气哪里能跟 Cornell 的号称 “CS 前五” 相比啊？Indiana 的 offer 晚来了几天。当收到 Indiana 的 offer 时，我已经接受了 Cornell. Hofstadter 很惊讶也很失望，因为他以为我一定会做他的学生，可是听说我接受了 Cornell 的 offer, 他也不知道该怎么办。我只隐约的记得他告诉我，学校的排名并不是最重要的东西.....

名气和权威的力量是如此之大，它让我不去选择真正欣赏我并且能给我真知的人。有时候回想起来，我当时真的是在寻找真知吗？我明白什么叫做真知吗？

Cornell 给了我什么呢？到现在想起来，它给我的东西恐怕只有教训，很多的教训。TA 的工作可不是那么好做的，基本就是苦力，你甚至会怀疑他们录取你就是为了利用你的廉价劳动力。我第一次做 TA 就是一个 200 多人在阶梯教室上的大课，教最基本的 Java 编程。虽然有好几个 TA，但任务还是很繁重。讲课的人不是教授，而是专职的讲师。这种讲师一般得靠本科生的好评来谋生，所以虽然在学术上没什么真本事，对学生真可谓是点头哈腰，服务周到。这就苦了各位 TA 了，作业要你设计，还要设计得巧妙，要准备好标准答案，之后还要批作业，批得你头脑麻木，考试要监考，之后还要批试卷。每周还得抽好几个小时来做 office hour, 给学生答疑。然后你还有自己要上的课，自己的作业，自己的考试。每当考试的时候都很紧张，因为你得准备自己的考试，还要为学生的考试多做很多工作。

如果真的学到了东西，这么辛苦也许还值得，可是那些教授真的是想教会你吗？有人打了个比方，说 Cornell 说要教你游泳，就把你推到水池里，任你自己扑腾。当你就要扑腾上岸时，他在你头上用榔头一砸，然后继续等你上岸。当你再次快要扑腾上岸时，他又举起一块大石头扔到你头上，这样你就可以死了，可是 Cornell 仍然等着你游上岸..... 这就是对我在 Cornell 的经历的非常确切的比喻。

我在一篇老的博文里面提到过，Cornell 的学生，包括博士生，一上课就抄笔记，一天到晚都在赶作业。可其实 Cornell 不只是爱抄笔记的学生的天堂，而且是崇拜权威者的天堂。即使你不是那么的崇拜权威，你不可避免的会被一群像朝圣者一样的人围在中间，在你耳边谈论某某人多么多么的牛。不管你向同学打听哪一个教授，得到的回答总是：“哇，他很牛的！” 然后你就去上了他的几节课，觉得不咋的嘛，可是人家就说那是因为你不理解他的价值。这种气氛我好像在另一个地方感觉到过呢？啊对了，那是在 Google. 这样的气氛也许并不是偶然，Cornell 的大部分 PhD 同学当时的最大愿望，就是毕业后能去 Google 工作。当然，后来 Facebook 上升成为了他们的首选。值得一提的是，Indiana 其实是更有个性的地方。我在 Indiana 的同学们一般都把去 Google 工作作为最后的选择之一。有一次一个刚来不久的学生问，如何才能进入 Google 工作？有个老教授说，那个容易，Google 招收任何能做出他们题目的人！

Cornell 的研究可以用“与时俱进”来形容，什么热门搞什么。当时 Facebook 和社交网络正在“崛起”，所以系里最热门的一个教授就是研究社交网络的。我去听过他几堂课，他用最容易的图论算法分析一些社交网络数据，然后得出一些“理论”。其中好些结论实在太显然了，我觉得街上的卖菜大妈都能猜到，还不如研究星际争霸来得有意思点。可是 Facebook 名气之大，跟着这位教授必然有出路啦，再加上有人在耳边煽风点火，所以有好多的学生为做他的 PhD 挤破了头皮，被刷下来的就只好另投门路了。每次新来一个教授都会被吹捧上天，说是多么多么的聪明，甚至称为天才。然后就有一群的人去上他的课，试图做他的学生。结果人家每节课都是背对学生面朝黑板，喃喃自语，写下一堆堆的公式和证明，一堂课总共就没回过几次头。下面的人当然是狂抄笔记，有的人甚至带着录音笔，生怕漏掉一句话。上这样的课还不如干脆把板书打印出来让大家自己回家看。人多了竞争也就难免了。上课的同学们就开始勾心斗角，三国演义的战术都拿出来了。作业做不出来就来找你讨论，等你想讨论了就说自己也没做出来。没听懂偏要故作点头状，显得听懂了，让你觉得有压力。自己越是喜欢的教授就越是说他不咋的，

扯淡，然后就自己去跟他。自己不喜欢的教授就告诉你他真是厉害啊，只可惜人家不要我。直到两年后我离开 Cornell 之前，还有好些同学因为没找到教授而焦头烂额。因为两年内没有找到导师的 PhD 学生，基本上等于必须退学。

当我离开 Cornell 之后，有一位国内的学生给我发 email 套磁（从系里主页上找到我的地址），问我 Cornell 情况如何。我告诉他我都已经走人了，并且告诉了他我的感觉，一天到晚抄笔记赶作业之类的。然后又问我一个刚毕业的 PhD 的情况，我说他水平不咋的，博士论文我看过了，很扯淡，解决一个根本不存在的问题。他对我说的话有点惊讶，但还是将信将疑。为了确保万无一失，他在 visiting day 的时候专程去 Cornell 考察了一下。回去又给我 email，说见到好多牛人啊，大开眼界，哪里像你说的那么不堪。还说跟那位 PhD 的导师谈过话，真是世界级的牛人那，他的博士论文也是世界一流的。我就无话可说了，仁者见仁，智者见智，随他去吧，哎。

结果两年之后，我又收到这位同学的 email，说他在 Cornell 还没找到导师，走投无路了，问我有没有办法转学。

2.5 图灵奖

说到这里应该有人会问这个问题， 我是不是也属于那种没找到导师走投无路的人。答案是， 对的， 我确实没有在 Cornell 找到可以做我导师的人。 然后我就猜到有人会说， 就知道王垠水平不行嘛， 没搞定导师， 被迫退学， 哈哈！ 可是事情其实没他们想象的那么简单。作为一个 PhD 学生，不仅必须精通学术，而且要懂得政治和行情。哦错了， 其实不精通学术也行的， 但是一定要懂得政治和行情！ 可是由于学生之间的窝里斗， 他们之间的信息互通程度， 是没法和教授之间的信息互通程度相比的。 这就造成了“学生阶级”在这场信息战上的劣势， 总是被动的被教授挑选， 而不能有效地挑选适合自己的教授。

进入 Cornell 之后我上了一门程序语言的课，就开始对这些东西入迷。可是由于“与时俱进”，Cornell 的研究方向并不是那么平衡的发展的，其实是很畸形的发展。程序语言领域的专家们早已因为受到忽视而转移阵地， 剩下一群用纸和笔做扯淡理论的。 说实话，在历史上程序语言方向曾经是 Cornell 的强项，出现了一些很厉害的成果。可是当我在 Cornell 的时候，只剩下两个名不见经传的教员，一个助理教授，一个副教授。其实 Robert Constable 也在那里，可惜的是他做了 dean 之后已经没空理学生了，以至于我两年之后都不知道这个人的存在。我当时也不知道 Cornell 有过这段历史，看不到它的研究重心的移动趋势。

我不喜欢那个副教授搞的项目，大部分是在 Java 上面加上一些函数式语言早就有的功能。可是人家做的是热门语言，所以拉得到资金，备受系里亲睐，他的学生们也比较趾高气昂。初次见面的时候，我跟他的一个学生说了我的一个想法，他说：“你那也能叫研究吗？待会儿我给你看看什么是真正的研究！” 其实那只是我的一个微不足道的想法，我也没说那是研究啊。只是随便聊一下而已就这么激动 -_- 何况你们那些 Java 的东西能算是研究？我是不可能跟那样的人合作的， 所以我就跟那个助理教授做了一点静态分析的项目。当然我们分析的也不是什么好东西，是用 Fortran 写的 MPI 程序。不过说实话，那个助理教授其实挺有点真知灼见，他有几句话现在仍然在指引我，防止我误入歧途。其中一句话是针对我对 π-calculus 的盲目崇拜 说的：“那些理论其实不管用的。最好是针对自己的问题，自己动脑筋想。” 他也是很谦虚很善良的人，可是好人不一定有好报的。后来他没有拿到 tenure 职位，不得不离开 Cornell 加入了工业界，而我就失去了最后一个有可能在程序语言方向做我的导师的人。

没办法，我就开始探索其它相关领域的教授，比如做数据库的，做系统的，看他们对相关的语言设计是否感兴趣。可惜他们都不感兴趣，而且告诉我程序语言领域太狭窄了。我当时还将信将疑，甚至附和他们的说法，可是现在我断定他们都是一知半解胡说八道。如果这些人虚心向程序语言专家请教，现在数据库和操作系统的设计也不会那么垃圾，关系式，SQL，NoSQL，..... 一个比一个扯淡。没有办法，我就开始探索其他的方向，开始了解图形学和数值分析等东西，进展很不错。可是终究我还是发现，我不喜欢图形学和数值分析所用的语言。我想制造出更好的程序语言来解决这些问题。可是跟教授们谈这些想法的时候就感觉是在对牛弹琴，他们完全不能理解。后来我发现，教授们貌似不喜欢有自己想法的学生，他们更希望找到愿意“打下手”的学生，帮助实现他们自己的想法。

这就让我走到了跟那位向我打听 Cornell 情况的同学差不多的局面，真是心里有许多的苦却没有人可以理解。这时候我想到了系里的一些德高望重的教授，比如得过图灵奖的人，也许这些顶级的大牛会给我指出方向。于是我就联系到一位图灵奖得主，说想找他聊聊。我说我感兴趣的东西 Cornell 貌似并不重视和发展。Cornell 的校训是 “any person, any study”, 而我想 study 的东西却得不到支持。最后我谈了一下我对 Cornell 的总体感受。我说我觉得大家上课死记硬背，不是很 intellectual, 我不是很确定学术界是否还保留有它原来的对智慧和真知的向往。

我很诚恳的告诉了他这些，只是希望得到一些建议。结果他不但没有理解任何一点，而且立马开始用质问的语气问我，你成绩怎么样？考试都通过了没有？哎，说白了就是想搞清楚你是不是成绩不好没人要。怎么就跟高中教导主任一样。于是乎那次谈话就这样不了了之。可是没有想到，这次谈话就造成了我最后的离别。在学生们互相之间勾心斗角，不通信息的同时，系里的教授们其实背后都是“通气”的。他们根本不懂得如何教学，就知道拿作业和考试往学生头上砸，幸存下来的就各自挑去做徒弟，挨不住的就打发掉。这算盘打得真是妙啊。我也不知道他们是什么机制，每个学生对哪些教授感兴趣，表现如何，他们貌似都了如指掌，貌似背后有个什么情报网。然后系里的教授们不知道怎么的，仿佛就都知道有这样一个不知趣的学生，居然敢说学术界的坏话！

大地震前夕的天空总是异常的美。我竟然在过道里看到那位图灵奖教授对我点头致意并且微笑，以前做 TA 时把我呼来唤去还横竖不满意的教授也对我笑脸相迎。我仿佛觉得那一席话打动了那位德高望重的教授，再加上在图形学和数值计算的扎实进展，也许我的学术生涯有了转机。可是，我那一次真正的领悟了什么叫做所谓的“笑里藏刀”。

由于那个学期上的图形学还有矩阵计算的课成绩都不错，我心想应该能找这两门课的授课教授的其中一个做导师吧。再加上那些貌似友好的笑容..... 所以没想很多，居然过了一个非常快乐的寒假。没有任何前兆，没有任何直接的通知（email, 电话），一封纸信不知道是什么时候默默地进到了我在系里的“信箱”—一个我基本上从来不看的，系里用来塞广告信息的信夹子里，直到下一个学期开始的时候（2月份）我才发现。信是系主任写的，大概就是说，由于你的表现，我们觉得 Cornell 不是适合你的地方.....

说得对，我也觉得 Cornell 不适合我。我本来就有想走的意思，可我一般呆在一个地方就懒得动。如果你们早一点告诉我这个，比如12月以前，我还可以申请转学到其它学校。可是都 2 月份了才收到这样的东西，Cornell 啊 Cornell, 你让我现在怎么办？我想我可以说你不仁不义吧？

在这个万分窘迫的时候，我想起了曾经关心过我却又很失望的 Hofstadter. 我告诉他我在 Cornell 很不开心，我很想研究程序语言，可是 Cornell 不理解也不在乎这个领域。他回信说，没有关系，你能找到自己喜欢的东西就应该去追寻它。Indiana 的 Dan Friedman 正好是做程序语言的，你可以联系他，就说是我介绍你去的。

于是给 Friedman 发了 email, 很快得到了回信说："Yin, 两年前我们都看过你的材料，我们觉得你是非常出众的学生，可惜你最后没有选择我们。你要明白，人生最重要的事情不是名利，而是找到你愿意合作的人。你的材料都还在我们这里。现在招生已经快结束了，但是我会把你的材料提交给招生委员会，让他们破例再次考虑你的申请." 我和 Dan Friedman 的故事就从这里开始了。

我在 Cornell 的遭遇貌似不可告人的耻辱和秘密，然而我今天却可以把它公之于世，因为 Cornell 不再有任何资格来评价我。依靠自己的努力和 Indiana 的老师们的培养，我的水平已经超越了 Cornell 计算机系的大部分教授。现在我觉得自己就像那个到 Cornell 学"游泳精髓"人，本来就是会游泳的，可是每到岸边 Cornell 就搬起大石头来砸我，还说我不会游。于是我钻到水底下钻了一个洞，把水放干。

由于曾经与多位图灵奖得主发生不大愉快的遭遇，再加上在自己的研究中多次受到其它图灵奖得主的理论的误导，而且许多位图灵奖得主最主要的贡献仍然在给软件行业带来混乱，图灵奖这个被许多计算机学生膜拜的神物，其实在我心里已经没有任何效力了。很多人可能对此难以想象，可是对图灵奖是这种态度的不只我一个人。我认识的几乎所有程序语言专家几乎都不拿图灵奖当回事，而且其中很多人甚至不拿图灵本人当回事，觉得他设计了一些非常丑陋的东西。虽然我现在觉得图灵的研究成果确实有一定价值，但由于上面的原因，拿图灵奖来开玩笑还是成为了我的家常便饭。我甚至觉得 ACM 应该停发这个奖，因为它是一种非常虚幻和政治的东西。每当人们谈起这些"大奖"煞有介事的时候，就让我看到了他们的愚昧。

2.6 常青藤联盟和"世界一流大学"

我在 Cornell 的经历应该不是偶然，不是因为我比较特殊。跟我同时进入 Cornell 的博士生有好几个没有拿学位就离开了。其中有一个是非常聪明的少年班，18岁就读 PhD 了，我根本听不懂的理论课他还能拿 A. 可是四年后他退学去了 Facebook, 说真是太难毕业了，神马都是扯淡。有些本科生也告诉我类似的经历，说被一个叫做"笑面虎"的教授"整了"。Cornell 的自杀率居美国大学前列。离开以后的有一天，忽然看到新闻报道说一周之内有三个 Cornell 学生从瀑布旁边的那座桥跳下去，结果派了警察在桥上日夜巡逻。我觉得自己在 Cornell 所感受到的压力确实超乎想象，是有可能把人逼上绝路的。现在回想起来真是可笑，因为下意识里在乎权威和名气，我给予了一群根本没有资格来教育我的人莫大的权力，让他们可以向我施加无端的压力。

应该指出，这种现象应该不是 Cornell 所特有的。我对清华，还有 Princeton, Harvard, MIT, Stanford, Berkeley, CMU 等学校的学生都有了解。这些所谓的 "世界一流大学" 或者 "世界一流大学 wannabee" 差不多都是类似的气氛。你冲着它们的名气和"关系网"挤破了头皮进去，然后就每天有人在你耳边对其它人感叹：哇，他好牛啊！发了好多 paper, 还得了 XX 奖。跟参加传销大会似的，让你怀疑这些人还有没有自尊。然后就是填鸭式的教育，无止境的作业和考试，让你感觉他们不是在"教育"你，而是在"筛选"你。这种筛选总是筛掉最差的，但也筛掉最好的。因为最好的学生能意识到你在干什么，他们不给你筛选他们的机会。一旦发现其实没学到东西，中途就辍学出去创业了。所以剩下来的就是最一般的，循规蹈矩听话的。在这样的环境里，你感觉不到真正的智慧和真知的存在。GRE 考试所鼓吹的什么 "批判性思维" (critical thinking) 在美国大学里其实是相当缺乏的。学生们只不过是在被培训成为某些其他人的工具，他们具有固定的思维定势，像是一个模子倒出来的。他们不是真正的创造者和开拓者。

人们在这些大学里的时候都是差不多感受的，可是一旦他们出来了，就会对此绝口不提。自己身上挂着这些学校的镀金牌子，怎么能砸了自己的品牌，长别人的威风？所以每当我批判 Cornell 就有些以前的同学一脸的着急相，好像自己没有吃过那苦头一样。

2.7 程序语言专家

虽然我在 Indiana 得到了思想的自由，但这种自由其实是以孤独为代价的。我并不是一个自高自大不合群的人，但是我不喜欢跟一群像追星族一样的人在一起。应该说在 Indiana 的日子里，权威主义的影子也是经常出现的。Indiana 学生们的权威比较特殊一点，不然就是 Dan Friedman, 不然就是 Kent Dybvig. Friedman 的身边总是围绕着一群自认为是天才的本科生，喜欢拍他的马屁，喜欢在人面前炫耀。博士生们开始时貌似还比较酷，可是后来发现其实也有很多类似现象，急于表现自己，越是研究能力弱的人越是爱表现。所以你就发现有人开头为了混进这个圈子拍你的马屁，过了两年就开始自高自大，而且经常想这样来压倒你："Kent 说过....." 我很尊敬 Dan 和 Kent, 但我其实在很多方面已经超越了他们。我看到他们的一些思维方式并不是那么的正确，我也从来不引用他们的话作为理论依据。对权威的崇拜其实显示了一个人心理的弱小。如果你对自己有信心，有自己的想法和判断力，又何必抬出个名人来压制别人呢？

在我自己心里毫无疑问的是，我是 Indiana 最厉害的程序语言（PL）学生。由于我不断地动手尝试新的想法，所以几乎没有任何其他人的研究逃脱过我的探索。我从来不记录自己的半成品和失败（因为太多了），而且我对自己的标准异常的高，所以我经常看到有人做演讲或者写论文，里面其实是我很久以前尝试过又抛弃了的想法。有时候我去听别人的演讲，就会立即看出破绽，问一些演讲者答不出来的问题。其实很多时候我只是怀疑自己，我试图给那些想法再一次的机会来证明它们的价值，而且问得相当委婉，但那样的问题仍然是不受欢迎的，所以同学们甚至一些助理教授看到我在场都是心惊胆战的。吃饭的时候我也不喜欢旁边的人讨论问题，因为他们经常显示出对理论提出者的膜拜心理，而且煞有介事，可惜那些经常是我早就知道不管用的理论。他们有时候其实也知道那些是扯淡的，但却又怕我捅破这窗户纸，所以就像鸵鸟一样把头埋在沙子下面。

我也想合群一点，但是屡试不爽，所以后来我就基本是孤立的做自己的研究了。最开头是不得已，但后来就越来越喜欢独自一人。这是不可避免的，因为创造力和孤独几乎是双胞胎。因为免去了跟人讨论的时间，我有了大把的时间来做自己的探索。然后我才发现当年期望的那种 common room 其实没什么用，因为那里根本不会有人理解你在说什么。现在即使有这样的地方我也不会去了。

我从一开始进入 Indiana 就没想过要拿博士学位，我只是在玩弄这个系统以达到我求知的目的。所以除非危及到我的存在，我把学校对学位的各种要求都抛到了九霄云外。给教授做 RA 几乎总是被要求研究各种毫无前途的东西，与我自己的思考相冲突，所以我后来干脆都做 TA 了。虽然累点，但不怎么费脑力。其结果是，在短短的一两年时间之内，我利用自己抠出来的时间，独自摸索出了这个领域大部分的理论。我经常不看书不看论文，在一个星期之内解决别人十多年才完成的研究。让人惊讶的应该不是我有多么聪明，而是这些研究者们十年来到底在干什么。我从来不认为自己比别人聪明，我只是觉得很多人的脑子被禁锢了而已。我有非常简单的头脑，我看不懂复杂的公式，听不懂高深的术语。可正是因为这一点，让我脱离了已有理论的困扰。

可以说， 这个领域在过去一个多世纪的研究， 很少有逃脱过我的洞察力和直觉的。这些研究最早可以追溯到 1870 年代。我一般很少看论文，因为自己想清楚一个问题其实花不了那么多时间的。 看别人的论文一般都枯燥乏味， 所以与其花那么多时间读论文还不如自己思考。 当我看论文的时候， 一般是想搞清楚自己琢磨出来的问题有没有人已经研究过了， 所以很多论文只需要扫一下就够了。 我看到一个东西一般很快就会知道它到底会不会管用。 我经常发现一些被认为很艰深的理论其实是在解决根本不存在的问题，甚至是在制造问题，而真正的问题却没有得到有效的解决。很多问题其实是权威的阴的，它让人们不敢否认这些大牛思想的价值，不敢揭穿它们，抛弃它们，甚至想让自己寄生在它们上面， 所以很多的时间花在了解决一些历史遗留问题， 而不是真正的问题。这就是为什么我的英文 blog 标题叫做 “Surely I Am Joking”, 因为它记录了一些我认为根本不存在，或者是人为造成的问题。

2.8 逻辑学家

批评 PL 领域的问题并不意味着其它领域就好一些。 恰恰相反， 我认为做系统和数据库的领域有更大的权威崇拜和扯淡的成分。 有时候程序语言专家看起来很明显的问题，做数据库和操作系统的人却看不到，扯来扯去扯不清楚，还自以为是的认为 PL 的东西他们都懂。

程序语言的理论是计算机科学的精髓所在， 可是程序语言专家有他们自己的问题：他们膜拜逻辑学家。几乎每一篇 PL 领域的论文，至少有一页纸里面排列着天罡北斗阵一样的稀奇古怪的逻辑符号， 而它们表示的其实不过是一些可以用程序语言轻松做出来的解释器和数据结构。有人（比如 Kent Dybivg）早就发现了这个规律，所以写了一些工具，可以把程序语言自动转换成 LaTeX 格式的逻辑公式，用以对付论文的写作。

有人觉得那些公式有“数学的美感”，可是它们其实是挺有毛病的设计。如果你看看现代逻辑学鼻祖 Gottlob Frege 的原著，就会发现其实最早的时候逻辑学不是用公式表示的。Frege 那篇开创性的论文 Begriffsschrift 里面全都是像电路图一样的图片，只有 20 多页， 而且非常容易读懂。 不知道是哪一个后辈把电路图改成了一些稀奇古怪的符号。其实他的目的是用符号来表示那些电路图， 结果到后来徒孙们以为那些符号就是祖传秘籍的精髓， 忘记了那些符号背后的电路图， 所以导致了今天的混乱局面。 看完了 Frege 的论文，我再一次领悟到了之前那句话：跟真正的大师学习，而不是跟他们的徒弟。

ACM SIGPLAN 的主席 Philip Wadler 有一次写了一篇论文介绍 Curry-Howard corresponce，里面提到，好的点子逻辑学家总是比我们先想到。可是他却没有发现，其实程序语言的能力已经大大超越了数理逻辑， 数理逻辑那些公式里面的 bug 其实不少。因为逻辑学家们不用机器帮助进行推理，有些问题搞了一百多年都搞不清楚是怎么回事，然后就弄出一些特殊情况和补丁来。有了一堆逻辑“定理”，却又不能确信它们是正确的，而且存在悖论一类无厘头的东西， 所以又掰出一些 model theory 之类的东西来验证它们的正确性。 逻辑学家们折腾了一百多年都是在折腾类似的事情， 却没怀疑过老祖宗的设计。 我之前提到的 Hindley-Milner 系统的问题， 很大部分原因就在于它所使用的逻辑里面其实有一个根本性的误解。 简言之， 就是把全称量词 ∀ 随意乱放， 导致输入与输出关系混乱。这也就是我为什么不喜欢 Haskell 和 OCaml 的最主要原因。

现在最热门的逻辑学家莫过于 Per Martin-Löf。他的类型理论 Martin-Löf Type Theory 被很多 PL 人奉为神圣。我一直没有搞清楚这个类型理论有什么特别，直到有一天我把 Martin-Löf 1980 年的那篇论文（其实是演讲稿）拿出来看了一遍。然后我发现他通篇本质上就是在讲一个 partial evaluator 要怎么写，而我早就自己写过 partial evaluator。 其实并不是特别神奇的东西， 只需要在普通解释器里面改一两行代码就行，可是有人（比如 Neil Jones）却为此写出了 400 多页的书和大量的论文。

之前提到的 Curry-Howard corresponce 也被很多人奉为神圣，它来自数学家 Haskell Curry 和逻辑学家 W.A. Howard 的一些早期发现。他们发现有些程序和定理的证明之间有对应的关系。然后就有 PL 专家开始走火入魔，说“程序就是证明，程序的类型就是定理”。可是他们却没有发现这个说法没法解释操作系统这种程序，因为它被设计为永远不停地运行，所以不能满足一个证明所具有的基本特征。而且很多程序被设计出来根本就不是要证明什么定理，它们是被设计来帮人做事情的。所以我觉得“程序就是证明”是很牵强附会的说法，你不能因为有的程序可以用来证明数学定理，就认为所有的程序都是某个定理的证明啊！把那句话反过来，说成“证明就是程序”还差不多。

但从以上的发现，我很高兴的看到了自己作为一个程序员的价值。很多人瞧不起程序员，把他们蔑称为“码农”，可是程序如果写好了，其实比起那些高深的逻辑学家和哲学家还要强，因为程序语言其实比数理逻辑还要强。有一位数学家说得好：为了真正深入的理解一个东西，你应该把它写成程序。还有人说，编程只是一门失传的艺术的别名，这门艺术叫做“思考”。我觉得很在理。

2.9 再见了，权威们

几经颠簸的求学生涯，让我获得了异常强大的力量。我的力量不仅来自于老师们的教诲，而且在于我自己不懈的追求，因为机会只亲睐有准备的头脑。

曾经 Knuth 是我心中唯一的权威，后来我又屈服于 Cornell 和常青藤联盟的权威和名气。在一而再再而三的上当受骗之后，我终于把所有的权威们从我的脑子里轰了下去。也许有时候轰得太猛烈了一些，但总的说来是有好处的。不再是我心目中的权威并不等于我鄙视他们或者不尊敬他们。我只是获得了不用膜拜他们，不用跟一群人瞎起哄的自由。我不尊敬的人都是一些自视过高的人或者他们的跟屁虫。一般来说，权威们在我的脑子里失去的只是他们在很多其他人脑子里的那种被膜拜的地位，那种你可以用 "XX 人说过....." 来压倒理性分析的地位。现在他们在我心目中是一群普通的，由蛋白质形成的生物，有好心肠或者坏心眼的，高傲，谦虚或者虚伪的人。我不会自讨苦吃，他们的想法如果真的好，我当然要拿来用，但是没有任何人的东西我是不加批判全盘接受的。我深深地知道接受错误想法的危害性，所以我也希望大家都具有批判的思维，不要盲目的接受我说的话。我不喜欢“大神”或者“牛人”这种称呼，我也反感那种自称膜拜我的人，因为正是这种人让权威主义现在横行于世。

美国的权威主义胜于欧洲，但也不是每个人都那么的崇拜权威，而中国才是权威主义的重灾区。像 “图灵奖得主 XX” 这样的称呼，恐怕只有在中国才见得到。所以我希望国内的同学们，不要盲目的崇拜国外的所谓“大师”，“牛校”或者“牛公司”。祝你们早日消灭掉心里的各种权威以及对他们的畏惧心理，认识到自己的价值和力量。

2.10 后记（关于 IU）

有些人看了我的文章介绍在 IU 的经历，告诉我他们申请了 IU. 我觉得有必要免责声明一下：我没想到，也不希望有人因为我的文章而去 IU, YMMV (your mileage may vary). 由于我有所准备，所以对于 Friedman 的教学如鱼得水。很多同学也说学到很多，可是有一些其他人告诉我他们觉得 Friedman 的课他们听起来很吃力，只能说是勉强过关。而且我只介绍了 IU 好的方面，却把不大好的地方一笔带过了。我在 IU 也有很艰难的时候。

现在的情况是 Kent Dybvig 已经离开了 IU，加入了 Cisco. 他的公司 Cadence Research Systems 和 Chez Scheme 也并入了 Cisco。Dan Friedman 由于年纪原因说不准还带不带学生。最近引进了一些貌似不错的助理教授，但是我跟他们都不熟。我的经验是助理教授一般都会为了研究资金，为了升为正教授而做一些身不由己的事情。其他的 CS 方向我都说不准 IU 是什么水平，所以还请同学们自己斟酌。我可以毫无疑问的一点是，IU 有非常美丽的校园，大大的超过清华, 北大, Cornell, Stanford, MIT.

3 对博士学位说永别

经过深思熟虑之后，我决定再次“抛弃”我的博士学位。这是我第三次决定离开博士学位，也应该是最后一次了。这应该不是什么惊人的消息，因为我虽然读博士 10 年了，可是我的目标从来就不是博士学位。我在寻找更重要的东西，而且那个东西已经被我找到了。所以我的“博士生涯”其实完成了它的使命，基本上可以圆满结束了。

如果你从我之前的博文判定我现在生活在我所向往的环境中，那么你就误会了。我学到了我想要的东西，但是却发现学术界不再是我向往的地方。相反，它阻碍了我的前进。很显然，博士学位这个东西其实已经被学校和学术界作为利用廉价劳动力的“无形枷锁”。你想要“博士”的头衔，那么就廉价给我们干活吧，能出论文的就出论文，能写代码的就写代码。我根本不需要“博士”这个头衔来显示自己的价值，所以我抛弃学位，离开学校，离开学术界，是一点都不心痛的。

3.1 思想的监狱

如果你以为学术界意味着思想的解放，对真知的无私分享，那么你就错了。涉猎不深的人往往有一种美好的幻觉，觉得老师都是在无私的传授知识。可是等你深入到一定程度，就会发现其实没有人对于知识是无私的。这对世界顶尖的科学家们也不例外，他们“分享”给你的东西，往往是一堆琢磨不透的符号和公式，他们提出一堆可怕的术语来吓唬你。他们告诉你的只是思维的“结果”，而不是思维的“方法”（也就是所谓“直觉”）。等你通过自己的独立思考得到同样的东西，却发现这些大师们其实一直把直觉隐藏在很深的地方，故意的让你知其然而不知其所以然。他们甚至会诋毁直觉本身的价值，试图让你相信他们想出那些公式没有通过使用直觉，试图让你相信只有那些吓人的公式，定理和证明才是可靠的，而直觉是不可靠的。可是真正的直觉却是非常强大的，只要得到了它，你就可以完全的理解那些公式，而且会知道它们是怎么来的，甚至发现里面存在的问题，想出新的公式。

然后你就会恍然的发现，你曾经认为的“思想的天堂”，其实是“思想的监狱”。你会发现你心目中的很多大师们其实不是真正的科学家，而是政治高手。你会发现身边有很多人其实是故意在你面前提起一些术语，以此来显示自己的“高深”。当你直言不讳的道出这些东西背后的秘密，他们就不说话了，然后就对你怀恨在心，希望你早点消失。当你给那些提出这些术语的大牛们发 email, 试图核实自己发现的这些术语背后的直觉，他们会很快的停止回复你的邮件。当你试图从教授们口中得到这种直觉，他们会不耐烦的对你说：“你问这种问题有什么意义吗？这东西就是这样的！”这说明了什么呢？这说明他们害怕了。他们害怕自己的地位受到威胁，他们害怕这种直觉一旦被大多数人掌握，他们就不再是高高在上的“学术权威”了。人们就会说：“那个东西其实不过是……”

这就是我这些年来所亲身经历的。我的同事们其实都不知道，他们所景仰的大师们提出的高深的理论，好些已经在我心目中被默默的“杀死”了。对我来说，它们不过是用一堆吓人的数学公式，翻来覆去的表达一些用几句话就可以说清楚的东西，而且它们好多其实已经被另外的东西所超越。所以大师们常做的一件事就是招收“门徒”，只要有人愿意做他们的接班人，把自己的工作“基于”他们的理论，他们提出的空洞的概念就可以一直存活下去，而他们就可以保持自己的地位。所以你就发现一些可笑的现象，本来一个新概念跟某个老概念没有关系，却被生拉硬拽在一起。本来一个概念可以被独立的理解，却被牵扯到一堆的老概念中，被搞得无比复杂。

这就是为什么我曾经提到，我经常用两个星期的时间就“灭掉”了某些领域长达 20 年的研究。这并不说明我是天才，这只是说明很多人在玩弄学术的把戏，而我看透了他们的把戏。看透了这些把戏并不能带来实际的效益，却可以让我自己节省下时间来解决真正重要的问题。但是我看不到学术界在这方面有任何改进的希望。所以，为了思想的自由，我不能生活在学术界。

3.2 论文的游戏

我曾经以为我的专业（程序语言）是计算机科学里面论文水分最少的地方，但是其实并不是这样的，天下乌鸦一般黑。程序语言专业的论文与其它专业唯一的区别是，这个领域的人玩的把戏更加巧妙。这些论文动不动就触及 Church, 图灵这样的人物提出的概念，所以就算里面没有任何新的内容，你早就被吓倒了，就别提看出里面的把戏。可是由于我看穿了一些核心概念的本质，所以经常浏览一些论文都发现其实没有任何新东西。这些论文有些甚至来自某些本领域众所皆知的“巨星”。由于他们地位显赫，这里我就不点名冒犯了。

很常见的一个套路就是，把一些很简单的“程序”用一堆“数学符号”改头换面写出来，把它们叫做“逻辑”或者“类型系统”。人们都崇拜数理逻辑，因为他们看不懂那一堆的符号和推理规则。可是经过自己的努力，我却看透了很多逻辑学家的把戏。我也可以玩这种把戏，我知道如何设计出新的逻辑。可是我也知道其实我们并不需要这些逻辑，所以我从来不为此发表论文。可是这样的关于“程序逻辑”的论文，仍然频繁的出现在最高档次的学术会议。一眼就知道他们在干什么，让我觉得非常无聊。

上个学期，我跟导师做了一个学期的研究，内容是关于“类型系统”。说是“跟导师”，但是其实他只起到绊脚石的作用。他不但没有真正的参与讨论，而且明显的对于我深入的发现有抵触情绪，并且不断的打击我。我用于“说服”他所使用的精力，比我用来研究的精力还要大好几倍。到头来我却发现，原来他根本没有听我在说什么！我总是发现一些复杂的类型系统的功能，要么就是完全不可能实用，要么就是可以用已有的更简单的方法实现。所以这些发现，虽然从实际意义上“杀死”了好几个长达几十年的领域，把它们归并为同一个非常简单的东西。也就是说，这些发现消灭了一些不必要存在的东西，而没有带来任何“新东西”。所以他总是对我说：“你的这些想法有什么用吗？”然后我才发现，原来他所要的，并不是深入的理解，而是别的东西。这个“别的东西”，当然就是论文和经费。很显然，导师们只是把学生作为可能给他带来论文和经费的人。他们更喜欢那种看不透事物本质的，对什么“新概念”都持手舞足蹈态度的学生。因为这些学生可以很快的写出一些看似高深却没有真东西的论文，然后从 NSF 等研究机构要到钱。

所以说，深入本质的认识，其实在学术界是不受尊重的。因为深入的认识往往是无比简单的。如此简单的东西，又怎么拉得到经费呢？

3.3 政治的泛滥

我可以不谦虚的说，我是这里最好的学生。我身边的同学，没有一个可以超越我所能做到的事情。而他们能做的事情，却没有一件是我不能做的。在 Dan Friedman 的两门高难度的课程都得到 A+, 在 R. Kent Dybvig 的 Scheme 编译器课程得到A+, 并且在某种程度上超越了他们。在两个星期之内，完全凭自己的思考写出正确的 CPS 变换（10多年的研究成果），以及 ANF 变换（我导师的最重要成果）。多次在两个星期之内，“灭掉”某些领域 20 多年的研究。对很多东西有自己独到深入的见解。在工程上有 Google 的那个项目，以及我自己的一些项目，.....

可是这所有的一切，都没有让我受到同行们的尊敬。相反，由于我喜欢说真话，他们恨我又怕我。他们都怕把自己的东西给我看，因为我总是能很快的道出它们的本质。那些精于玩弄政治的人，显然更加得到人们的赏识。我本来不想把这些“业绩”摆出来的，实际上这些事情我一直以来都没有在同学面前提起过。可是显然，在 Friedman 和 Dybvig 的课程上挣扎不堪的同学，在某些会议发过一点鸡毛蒜皮文章的同学，现在很是趾高气昂，他们当然希望别人都不知道我的优秀。

可是我没有告诉他们的实在是太多了。他们的论文里面的内容，其实好些是我几年前就得到了的结果，只不过我懒得为这些鸡毛蒜皮的东西写论文。记得有一次，几个 Friedman 课上的同学想做一个暑期项目，试图改进 miniKanren (Dan Friedman 设计的逻辑语言) 的效率。那个时候我已经重新实现了 miniKanren, 并且独立的加入了一些扩展的功能，比如 constraint logicprogramming. 我一听到他们这个计划，立即就告诉他们，miniKanren 所使用的 substitution 的数据结构是 associationlist, 查找时间是线性的，显然效率很低。使用另外一种“函数式数据结构”，比如函数式的平衡树就会好很多。可是当他们听到这些的时候，居然保持了一种可怕的沉默，完全把我的话忽略了，就像我并不存在于他们面前！一年多之后，他们发表了一篇论文，里面的基本内容就是我告诉他们的那些话。当然，署名里面没有我。我根本不在乎这么小的想法，我本来就觉得他们根本不应该为此写论文。可是一点谢意也没有表达，倒觉得是他们自己了不起，真是让人难以接受。

另外还有好几次类似的情况，我都不想说了。后来我才从一个同学口中得到一些真正的信息。他说，某些人喜欢在听到别人好的想法的时候，进行故意的打击，或者莫不关心的样子。等别人对他们自己的想法失去兴趣之后，他们却把他的想法拿出来署上自己的名字发表。这种事情就曾经发生在他的身上。他的导师一直打击他，以至于他写出来的论文两年都没有发表。等他决定离开之后，却发现自己的想法被导师和另外一个人发表了。没有署上他的名字。

这是对知识的公然的盗窃甚至抢劫！这也是我越来越不喜欢跟人讨论的原因，因为很多人得到那些想法，就会想玩弄一些花招把它们据为己有，以此出名，却连个谢字都不说一声。我看到这种现象不只存在于我的身边，而且存在于整个学术界。有这样的政治和勾心斗角，我留在这里还有什么意义呢？我在这里，以至于整个学术界，其实已经没有什么值得合作的人了。

3.4 何去何从

当然接下来我需要思考的是，我应该做什么。显然，我所学到的知识可以轻而易举的给我带来高薪的工作。可是我也知道，我知道的这些东西，其实归根结底就是那么几个小把戏。所以我更愿意探索更加广阔的世界，学会新的东西，找到真正值得合作的人，创造真正的价值。另外，我也会逐渐把我知道的这些“把戏”以直观而容易理解的方式公诸于众。

産業篇

1 我的第一次和最后一次 Hackathon 经历

在旧金山地区经常有一些叫做 “Hackathon” 的活动，吸引挺多人参加。我一直听说这个名字，可是一直不知道它到底是什么。我从来对竞赛式的活动不感兴趣，从来不参加 ACM, IOI, TopCoder 之类的竞赛。可是在 Voxer 工作的时候，一天看到有个大公司主办了一个叫做 “data science” 什么的活动，以为是个讲座或者交流会，又因为我将要做 data science 相关的工作，就想去了解一下。可是没想到，那成为了我的第一次 Hackathon 经历。

一进门就感觉这跟一般的 meetup 气氛很不一样。这大周末晚上的，清一色的爷们，没有一个女人，也没有笑声。而且里面的人说话都很奇怪，不正眼看人，有些好像怒目相向的样子，说出话来就像在查你户口。有几次有人问我是干什么的，我刚一开口，他们一句话不回，扭头就跟其他人说话去了。只有一个头发花白的工程师对我挺友好的，于是我们就聊起来。旁边有个华人工程师盯着一个 15 寸的 Retina Macbook, 后来也聊起来，开门见山就问我用什么语言，我也忘了我说什么了，只记得他很自豪的说自己用 JavaScript, 而且那是最高配置的 Macbook.

等坐到一个像教室一样的房间里面，我才发现这是一个 Hackathon 而不是一个讲座。是一个叫 Kaggle 的公司，联合了像 Neo Technology（产品是 Neo4j）之类的公司组织的。后来我发现，这个 Kaggle 是专门搞 data scientist 的“竞赛”的。我以前从来没听说过这公司，也不知道 data science 还有竞赛。

有点失望，但还是有点好奇，所以暂时没有离开。等大家都进了房间，台上一个人开始讲话。一开口我就震惊了：“你们都知道今天来这里是干什么的吗？！.....”我是第一次听到这样的开场白，比监考老师还要厉害一些。然后他就开始讲 Kaggle 的事情，貌似每个人都知道那是什么一样。言语之中不时地冒出像 “世界第三的 data scientist” 之类的词汇。哇，我第一次听说，原来科学家还有排名之说！

我开始后悔自己只瞄了一眼网站上的广告就来了，都没仔细看他们要干什么。从主持人的言语里我才了解到，这些人来到这里是为了一个竞赛。他们需要组队，在那个房间里待一天一夜，连续奋战之后提交他们的答案，为的是 500 美元的奖金和可以放在简历上的“荣誉”。从周五晚上通宵达旦到周六晚上，太不可思议了。题目我记不清楚了，貌似就是一个 CSV 文件里存着一些社交网络的数据，想要预测什么东西。

最后主持人说：“吃的，喝的，都在外面，保证你们有足够的卡路里。大家开始寻找合作伙伴吧！”然后旁边那位之前聊天的大叔就朝我微笑，貌似想找我做队友。我才窘迫的告诉他，我其实不知道这是一个 Hackathon，我以为这有一个 talk 就来了..... 然后就发现他脸色大变，仿佛之前浪费了宝贵的时间跟我废话似的，立即找其他人搭话去了。伤心难过落荒而逃之前，我还是客气的对他道了声晚安，结果得到一个很没好气的 "Good night!"

这件事已经过去好几个月了，最近发现 twitter 上有人这样说：

hackathon am not only an place to sell you dignity and time but now you can sell you soul too.

确实挺符合我的经历的。希望这就是我的最后一次 Hackathon 经历吧。下次不要再不小心参加类似活动了 :)

2 我和 Google 的故事

也许有人看见过我批判 Google 的那篇英文文章。它有一部分片面性，所以被我从英文博客上拿下来了。我一直在反思自己在 Google 的经历，也许现在用自己的母语，我可以得出一个准确一点的结论吧。

也许有人觉得作为一个读了这么多年的 PhD 去给别人做实习生（intern）是一种耻辱，但是我亲眼看到，从一些名校比如 Yale 毕业的 PhD，在 Google 混了好几年，也不过是过着差不多的生活。只不过做了 intern 之后我长了经验，知道了自己的价值，以后不至于落到同样的位置。

这里我就讲述一下我在 Google 的实习经历吧，也许对人有参考作用。

先说说我的项目是怎么开始的吧。当我加入的时候，我的老板 Steve Yegge 的小组试图制造一个跨语言的“服务式”的编程工具，叫做 Grok. 你可以把它想象成 Eclipse, 但是 Grok 的设计目标不只是像 Eclipse 那样检索和分析本机的某一种语言的代码，而是大规模的检索和分析 Google 的所有项目，所有语言，所有代码。这包括 Google 的 "四大语言": C++, Java, JavaScript, Python, 一些工具性的语言: Sawzall, protobuf 等, 还有一些 “build file” 和所有第三方的库。Grok 的初期设计目标是一个静态的代码索引服务，只要程序员点击任何一个变量或者函数名，就能“准确”的跳转到它定义的位置。动态的编辑功能稍后也在陆续加入。

这种检索不是像 ctags, etags 那种简单的正则表达式匹配，而是像 Eclipse 和 Visual Studio 那样的准确的“语义检索”，所以它必须真正的理解程序语言的语义。 在 Grok 诞生以前， 市面上和 Google 内部都没有一个工具能正确的支持所有“四大语言”， 所以我不得不说，Steve 的项目比起 Google 的其他更编程语言相关的项目，是相当先进的。

虽然 Grok 的技术含量很高，但是 Google 的管理层对东西的评价并不是看技术含量的，而是看你有多少 "影响力" (impact), 说白了也就是有多少用户。Google 当时本来 就只有不到一万个程序员，一个“内部编程工具”能有多少“用户”呢？所以 Grok 比起像 CodeSearch 一类利用正则表达式来查询程序的“低端”项目来说，在管理层心目中并不占任何优势。而且由于其它项目界面好看些，用户多些，Grok 随时有被取消的危险，这使得 Steve 心理压力很大。我就是在这个“危难关头”进入他们的小组的。 我当然没蠢到会自己进入这样一个组，但是 Steve 在电话面试时把一切都说得很美好的样子。当时小组里只有三个人：Steve 和另外两个组员。

当我开始的时候，Grok 小组已经初步完成了 Java 和 JavaScript 的检索模块。但是他们的检索并不是从头设计的，而是从 Eclipse (JDT) 和 JSCompiler 里面分别“挖取”了对 Java 和 JavaScript 语义检索的部分，修改之后插入到项目里的。Eclipse 的设计非常的不模块化，以至于项目进行了一年多，大家还在忙着解决它带来的各种 bug.

最开头的时候 Steve 给了我两个选择：检索 C++ 或者是 Python。我觉得 C++ 的设计太繁琐，所以就选择了看起来好一点的 Python. Steve 就让我去找一个好一点的开源的 Python IDE，然后把里面的语义检索部分挖出来插入到项目里面。可是在看过十个左右的 "Python IDE" 之后，我发现它们没有一个能够正确的“跳转到定义”。分析其 原因，是因为这些 IDE 基本上做的是正则表达式匹配，而完全不理解 Python 的语义。所以我对 Steve 说，我要自己从头写一个。但他反对这个提议，因为他觉得这是三个月的时间之内不可能完成的。不但是我不能，而且就算一个小组的高级程序员也不可能完成。就算完成了，他也不想“维护”这些代码。所以他宁愿让我去拿一个不怎么样的开源项目，因为这样“维护”的工作就转嫁到开源项目身上去了。

可是我很清楚的看到，这样一个语义检索，不过是一个抽象解释器 (abstract interpreter). 写解释器是我最在行的，所以我告诉他这是我可以完成的，而且由于设计上的简洁，我的代码的维护代价会比使用一个开源项目小很多。他没有说话。我同时也在进行一些内部培训，看一些视频，折腾 MapReduce 一类的内部工具教程，就这样过了一个星期。我隐约的感觉到 Steve 的不快，因为他不怎么说话了，可是我也没有太在意，仍然傻乎乎的到处凑热闹。到了周五的时候，Steve 下午很早就回家了。另一个组员还待在哪里，不时的叹气。我对她说：“Steve 是不是不高兴了？我知道我说话有点太自信，可能打击到他了。”她好像打满的气球被开了一个洞：“他怎么会被你打 击到？你知道他以前做的项目有多厉害吗？他是怕你做不出来。之前有一些 intern 设的目标太高，以至于到最后没有完成他们的项目。”于是她打开 Eclipse，把 JSCompiler 的代码给我看。“你知道我们以前一个类似的项目 JSCompiler，花了多少时间才完成吗？一个小组的人，四年的时间！”她打开其中一个文件，也就是处理符号 表的那个模块，说：“看这一个文件就有 9000 多行代码。你三个月能写出这么多代码吗？”我翻了一下白眼，搞笑似地说：“啊～ 怎么可能有 9000 多行？这些人真的知道怎么写这种代码吗.....”

后来具体的对话我忘记了，但是她说得那么神乎其神，确实给了我一些压力。再加上 Steve 那个闷声子，真是不好受。所以那个周末我没有出去玩，我下载了一个 Jython, 把它的 parser 文件 (ANTLR) 拿出来。自己设计了一个更简单的 AST 数据结构，把这个 parser 生成的 AST 转换成我的结构。然后就开始在上面写一个抽象解释器。由于 Java 的限制，我想出了一个更简洁的用 Java 实现解释器的方法，从而避免了使用繁琐的 visitor pattern. 一个周末之后，我做出了一个基本的原型。当然因为 Python 语言的复杂性，有很多细节的东西到后来才完全的实现。

等到星期一的时候，我告诉 Steve 我做了一个原型出来，而且因为我拿了 Jython 的 parser, 我们以后可以用这个理由把这代码 merge 回 Jython, 给他们提供功能，让他们帮我们维护代码，对两方都有好处。他居然一点也不高兴，把我叫到一个白板前面，板着脸说：“来，给我讲一下你打算怎么做。”我就画了一个 AST 的类关系图，在里 面每个类插入一个叫 interp 的方法，然后指出这个东西就是一个解释器。最后他豁然开朗了一样，说：“好。我相信你知道你在干什么了。就这样做吧。”

在 Google 的整个夏天我都觉得跟其他人没有共同语言。我感兴趣的东西他们一点都不了解，所以我也不想谈。我觉得不以为然的一些东西，却被他们捧上了天。也许就是所 谓“价值观”不一致。总体感觉就是过度“和谐”，见到什么都夸赞，没有人说真话，以至于你不知道到底什么好，什么不好。而且人们总是喜欢谈论一些人的显赫“地位”，以及传说他们如何的“牛”。比如，有一次几个人在谈论一个 Google 的“牛人”，说他做了一个多么了不起的项目，很快就升为了 Staff Software Engineer（"Staff " 是比 "Senior" 高一级的职位，Steve 就是个 Staff）。我去看了一下这项目，发现不过就是 JUnit 的 "C++ 版本"。JUnit 这东西技术含量本来就是相当低的，做这样 一个东西就能当 "Staff", 那我岂不是轻而易举就可以成为 "Principal" 了？哈哈。我心里这样想，但是没有说出来。

一个 Staff 就如此，谈到 Google 的两个创始人的 时候，有些人就简直是黑白不分了。对他们的各种武断的甚至不讲理的做法， 居然都津津乐道。 创始人在他们眼里俨然就跟皇帝一样， 他们做什么都是对的， 甚至有人以他们曾经的办公室在创始人的正下方为豪。这种浮夸和互相吹捧之风， 恐怕是在其它公司也少见的。 Google 要求员工们保持一种 "Googley" 的态度， 原来就是这样的态度， 过度 “正面”和“积极”。 美国所崇尚的“个人主义”和“批判性思维”，我在 Google 还真的没有见到过。

另一些时候， 我会遇到一些对某种语言或者技术有宗教情绪的人。有一次一个工程师坐到我面前， 像是在面试我一样，正儿八经的开始自我介绍，后来我们就谈到 C++。我说 C++ 设计实在是太繁琐了，其实很多简单的语言效率并不比 C++ 低，C++ 最近其实在向其它高级语言学一些东西..... 后来这人就不说话了。那天以后我就发现跟他打招呼 他都不理了。后来我才发现，在 Google 是不可以指出某种语言，特别是 C++ 的缺点的。C++ 在 Google 的势力之大，连 Java 都只能算二流货色。

最搞笑的其实是 Google 总喜欢故弄玄虚，把一些微不足道的东西说得很玄乎。很多文档，视频，活动都挂着 "Google Confidential" 的标签。等你去看了，却发现其实是 众所皆知的东西，没有什么机密可言。可是大部分的实习生们却有一种受宠若惊的感觉， 以至于产生优越感。 每个星期五， 都会有一个 "TGIF", 两个创始人会像主持人一样 组织一个大会。本来无可非议，但是总感觉气氛过于群情激昂了，有点像小学的时候升国旗开大会的感觉。好不容易大家聚在一起， 总是在听新闻发布， 不然就是谈工作进度 ，不然就是表彰某些人。总之，你总是感觉到虚荣心或者自尊心在受到某种挑拨，有一种传销公司大会的感觉。 真正大家轻轻松松一起玩的 party, 却非常稀少。 你总感觉 Google 在你的生活里无所不在。所以一些别的公司的人（比如我寄宿的房子主人）都在疑惑，Google 的员工到底有没有下班的时间。

我就是这样度过在 Google 的每一天，以至于后来我都不怎么在饭桌上吃饭了。自己把饭端到靠墙的吧台去吃，或者坐在“冰激凌吧”跟里面的厨师聊天，省得遇到一些高谈阔论的人无语。 我发现自己跟打扫卫生的大妈小妹们也谈得来， 她们也喜欢跟我说话。后来我发现有这种感觉的不只是我， 另外两个比较厉害的博士生也懒的在那边吃饭了。其中一个说他一个星期就把自己的项目做完了，然后假装仍然在做，免得又被增加任务。这就是所谓“能者多劳”吧。掌握了核心技术的人，往往会有比一般程序员大几十， 上百倍的效率，可是得到的“回报”却是更多的任务量和压力。

虽然 Steve “允许” 我自己从头做一个 Python 分析器，但是这却不是没有压力的。这种感觉就像是“皇帝的新装”里的织布工一样。我扬言自己会做出精美绝伦的布料， 皇帝的大臣们却看不见， 所以他们就相当的小心。 总是对我很敬畏的样子， 有时会来问候一下， 做得怎么样了。 可是一旦扯到深入的话题， 却又怕被看穿其实他们不懂很多东 西。因为我的教授们研究 Scheme, 所以 Steve 有时候也会很激动的表扬 Scheme, 或者类似 Scheme 的语言比如 Clojure. 这种奉承真的让我受不了，生搬来的术语都是错乱的。为什么程序语言总是引起这种宗教的态度，不是抵制就是膜拜？

有一次一个 Staff Software Engineer 来访。看我在做这个 Python 分析器，很鄙夷的样子，说：“你做那个东西干什么。Python 本来是没有类型的，怎么推导得出类型来？我倒希望 Java 的类型推导做得更好一些，不用手写很多类型。”显然他不知道什么是类型推导，他也不知道如何把 Java 的类型推导做得更好。很多人把自己的命运寄托在语言的设计者身上，自己没有能力去改进它们，所以他们才会对程序语言顶礼膜拜。

直到有一天，我才发现 Steve 为什么这么紧张。那天有另一个 “分舵” 的 director 来访。他给我们做了一个关于 “创新”（innovation）的演讲。基本内容就是说，技术 上的创新，如果吸引不到用户，那就不算什么创新，拉得到用户的东西才叫创新。

那天下午，这个 director 来到我们的办公室。表情严肃的 “审问” Steve: “你说你每天有 5000 个用户。可是 Google 总共还不到 10000 个程序员。你是怎么算的？你 把接受你的服务的那些下游项目的用户全都算进去了吧！” 唉， 想不到大名鼎鼎的 Steve Yegge 在这种皇帝的钦差大臣面前也只能唯唯诺诺。

我可以说，这个 Python 的东西，虽然不费我很多力气，但却是 Google 里很少有人可以做出来的。就算 Python 的创造者 Guido van Rossum 恐怕也玄。所以实际上我的这个东西在很大程度上拯救了这个濒临灭亡的项目，因为一旦 Grok 支持所有的 "Google 语言", 就会有很多人注意到这个东西，从而会有 “影响力”。这确实是后来发生的 事，我走了之后，Grok 开始通过 API 给很多项目提供服务，包括 CodeSearch.

Google 给我的那点工资， 其实是根本买不起这样的软件的。 你可以参考一下像 CodeSonar 之类“静态分析”软件的价格，一份基本上就是我三个月的工资。由于我上学想找点外快，让他们捡了一个便宜。 可是这种“上级领导”的压力居然也间接的传到了我身上，而且是以一种非常不尊重的方式。这种感觉就是，你做得再多再出色，你相对于 Google 的“大拿”们， 什么都不算。 这也许就是 Google 为什么雇佣 Dennis Ritchie, Brian Kernighan, Ken Thompson, Rob Pike, Guido van Rossum 等大牛吧 。因为它就可以说：“看我们 Google 有这些顶尖牛人，你算个什么，要不断努力！” Steve 不止一次的对我说：“你要为 Google 做出杰出的贡献啊！ Google 的东西总是最好的，你要做出 Google 一贯的品质来。你知道 Python 的创造者 Guido 也是 Google 的员工吗？我一定会在他面前给你美言几句。” 这种语气，我好像在几十年前 的中国听说过呢？“你要为祖国做出杰出的贡献！” 他也许以为我会受宠若惊，可是我心里却不是个滋味。

其它组员如果看我貌似心情比较轻松，也会不时的提一下：“这个做完了吗？如果这个做完了，你可以做那个。反正我们有的是事情给你做.....” 我心里其实在想，你知道 这东西的 “难度” 吗？符号表模块都要写 9000 行代码的人，你自己来做一下，看看一年之内你做得出来不。总之他们就是用这种奉承，利诱，竞争，加威胁的方式，想方设法让我多做事情。 可是我心里想的是， Google 老爹， 您就给了那么点钱， 您想买多少东西啊？

本来这系统能做出来就不错了，一个组员却一直催着我写 test. 她根本不明白，一个程序并不是写了测试就会是个好程序。 这个程序经过我多次的大规模修改甚至推翻重来 ，即使一早写了测试，那些测试也会很快作废。这种大公司给人灌输的 “test-driven” 编程方式，在这种创造性的程序设计里是根本就是行不通的。要写出这样一个系统， 必须全神贯注，深入到语言的本质。 而去写测试，往往会打乱原来的思路，所以测试应该是最后完成之后才写的。当我最后完成这个系统，可以大规模的处理 Python 代码的 时候，却听见从她的桌上传来一声沉闷的咆哮："WRITE--THE--TESTS---" 这真的非常的 Googley！

最后我顺利完成了整个项目，从头到尾都是我一个人设计实现的（除了 Jython 里的 parser）。现在它每天都会把 Google 所有的 Python 代码索引一遍。很多内部工具比如 CodeSearch 里面的 Python 文件上的链接，都是这东西做出来的。我所有的代码加起来才 4000 行。处理符号表的模块只有 600 行。我怎么也想不通为什么 JSCompiler 会有 9000 行来处理这么简单的东西，但是也许这就是为什么 JSCompiler 花费了四年时间。

所以这就是我对 Google 的印象。过度 “和谐” 的气氛，工作和生活不能明显区分，过度保密，能者多劳，官僚主义。Google 总是号称自己的工程师 "build things ground up", 但是看来真正能做到的人没有几个。反倒你真想要 “从头” 设计的时候，会受到相当大的阻力。

有好几次我都看到很不错的工程师被 Google 雇佣了之后就销声匿迹了一样，为 Google “默默奉献” 一生，不再有自己的发明创造。我感觉 Google 就是一个埋没人才的机器，而它的 “创造性” 的名声，却让越来越多的人才被埋没。主动找上门的人才被埋没了不说，还吞并其它公司，并且对他们施行同样的 "Google 文化", 埋没更多的人才。

所以我建议有志气的人们，选择公司的时候不要看一个公司多么强大，雇了多少牛人，而要看它是否真的赏识和尊重你，因为你自己才是最重要的。

3 我离开了 Coverity

在写这篇博文的时候，我已经不再是 Coverity 的员工了，我已经在今天下午向公司正式辞职。

走出公司的大门，我觉得一身的轻松。这是我几个月以来第一次感受到加州美丽的阳光，AT&T Park 到处是欢笑的人群，他们是来看巨人队的棒球赛的。我第一次发现他们的面庞是那么的美，那么的友善。湾里的海水也格外的蓝，水面上船帆招展，一幅恬静自然，其乐融融的景象。我就像是一个刚从 Alcatraz（恶魔岛）释放出来的囚犯。我已经很久没有欣赏过这样的风景了，虽然我每天都从这风景中走过。

进入 Coverity 之前，我就在 glassdoor（一个让员工评价自己公司的网站）上面看过给它的评价，只有 3.2 颗星，44% 的员工推荐朋友去那里工作。评价者们写到：“管理队伍非常不成熟”，“不重视自己的员工”，“高层总是互相打架”，“每个星期都有人莫名其妙的被炒鱿鱼”，“过劳工作，工资太低”，“工程师非常聪明，可是不受尊重”，“你不再是一个人，你是一个数字”，“对新人不友好”.....

可是6个月以前，我认定了 Coverity 拥有我想要探索的技术，所以尽管如此的恶评如潮，还是毅然的加入了这个公司。现在我如愿以偿了。Coverity 的产品里确实有几个超过我的点子，我很快的把它们都学过来了（虽然他们压根没教过我）。Coverity 实现了几个我设想中的点子，从而让我的眼光的正确性得到了免费的证明。然而很可惜的是，由于 Coverity 的工程师们的不虚心，他们没能从我这里学到任何东西，虽然我们相处合作很友好。

然而，glassdoor 的评价者们对公司管理层的每一条批评，也几乎都一一的兑现了。对于管理层的不满，以及对自己的身体和心理健康的考虑，是我离开 Coverity 的真正原因。

这恐怕是一个罕见的既有高科技，却又极其吝啬而压榨的公司。Coverity 工程师的水平都是高于普通程序员的（有多少人会设计静态分析软件呢？没有很多），好些工程师都有博士学位。可是这些极其聪明的工程师，却并没有得到他们应该得到的待遇和尊敬，他们过着非常不轻松的生活。他们的工资并不比其它公司的普通程序员的工资高。每个人的头顶上，都仿佛有一双眼睛在随时盯着，督促着你干活。你一天工作了多少个小时，每个任务的“估计时间”，你花在任务上的实际时间，全都使用一种叫做 Jira 的软件进行记录。开会时 manager 会不断地向你提醒 worst case time, best case time, 要你做“ "top performer"..... 仿佛你的价值就只在于完成任务所花的时间，你还要跟其他人竞争！一星期一大会，一天一小会，要你报告前一天完成了什么，今天准备做什么。仿佛生怕你就偷懒了。这就是他们所谓的 "Agile" 管理模式，其原理就像是操作系统一样，把人作为可以任意调用和替换的 “进程”，并且并发执行。很可惜，这种管理模式，造成了软件质量的低下，bug 多得不计其数，而且难以修复。

最令我惊奇的其实是 manager 的言语里随时随地透露出来隐约的 “威胁” 口气，仿佛随时都在质疑员工的工作态度和积极性，随时都在检查员工是否工作够了时间，随时都在琢磨要炒谁的鱿鱼（这样可以节省点开支）。这是极度的不自信，仿佛他们不相信有人真的愿意为他们工作，随时都在对员工察言观色，生怕一下子走人了没人来给他们修补 bug. 所以公司里总是感觉一种人人自危的气氛。感觉这怎么不像是一个高科技公司，而是麦当劳呢？比麦当劳还小气，一副斤斤计较的穷酸样。

这里没有 Google 那样漂亮的办公室，健身房，没有免费的午餐和晚餐，没有舒服的 bean bags, 连冰箱里的免费饮料都是最便宜最不健康的可乐和雪碧一类的..... 所以相对而言，Google 的环境实在是好太多了（虽然我不大可能再去 Google 工作）。喂，从 NASA，洛马，波音那里赚来的钱都到哪里去了啊？

我经常发现好几个工程师晚上工作到八，九点。一个同事因为住的远，6 点就冲去坐火车（Caltrain），可是过不久我就发现他屏幕的 VNC 在动，我能清晰地看到他在继续工作，直到很晚..... 呵呵，我为什么知道这些呢？因为我以前也工作到很晚。我工作到很晚是因为我觉得那代码里面还是能给我一定启发，而他工作到很晚是因为他想拿绿卡。哈哈，绿卡，我才不会为了绿卡累坏自己。

我知道自己的价值。我是唯一能够第一眼就看出 Coverity 的代码里的严重错误的人。我也知道，在弥补了我的 Python 分析器的几个缺点之后，我可以用很短的时间做出可以与 Coverity 匹敌的产品（如果我愿意的话）。当我向 manager 提到我可以做一个 open source 的 Python 分析器的时候，我收到的第一反应不是赞扬，而是 “提醒”（或者说叫威胁）。他说，如果你现在做这个东西，恐怕 Coverity 的法律部门就会来找你，称这个东西属于 Coverity。呵呵，Coverity 根本就不会做 Python 的静态分析，我做出来倒属于你了？

整个公司总是处于一种压抑的气氛之中，很少见到人们的笑脸。有少数的人总是嘻嘻哈哈，可是那些都是 HR, Sales, ... 我不觉得他们的笑声中存在真诚的喜悦。很假。

当我递交辞呈的时候，HR 对我说：“随便你到湾区哪一家公司，都是差不多的情况。”但我不相信这就是整个的 “软件行业”，否则软件行业就是在制造新的奴隶社会。我相信，世界上还是有很多与我志同道合的人。

4 代码进入闭源状态

我做出了一个可能让很多人遗憾的决定。从今天开始，我曾经开源放在 GitHub 上的代码，除了教育性质的代码，全部进入私有闭源状态。这些代码包括 PySonar2, RubySonar, ydiff 等等，它们已经从 GitHub 上消失。从今以后，除非用于教育目的，我将不再开源任何代码。当然，大家已经下载的那些代码，仍然可以按照开源许可证免费使用，然而最新的改进以及将来的新产品，将全部闭源。做出这个决定的大部分原因，是因为多次对人心的失望。

PySonar2 一度处于开源状态，使用宽松的 BSD 和 Apache 版权。PySonar 的用户包括了 Google, Sourcegraph, 还有其它几个我不能透露名字的，做专业代码管理工具的公司。很多人崇尚 BSD 这样的宽松版权，因为这样可以最大限度的传播代码。他们甚至把这作为了一种信仰，对于 GPL 这类严格限制商业用途的版权嗤之以鼻。甚至遮住眼睛对你说：“你的代码是 GPL 的，我不能看！看了之后写出一样的代码来，你会起诉我！” 然而多年的经历之后，我才发现 BSD 并不是好的开源版权，它其实会让代码的作者失去自由，而 GPL 才是真正保护软件及其作者的自由的。就像 Stallman 说的，自由软件（Free Software）这个词里的 "free", 是自由的意思，而不是免费的意思。直到今天，我才明白他这句话的真正含义。

为什么 BSD 版权会让人失去自由？这个故事要从 Sourcegraph 讲起.....

Sourcegraph 是一家做代码管理工具的公司。 他们初期的系统， 其实只是在 PySonar 之上做了一个简单的 web 包装。把 PySonar 分析出来的信息倒到数据库里面，然后通过 web 方式显示给用户。PySonar 本身早就有一个演示程序，可以生成互动的 HTML, 所以其实 Sourcegraph 能做的事情，我很容易就可以做到，只不过多一些杂活而已。Sourcegraph 并没有在 PySonar 之上增添很多的新东西，也无法做出 PySonar 这样的核心技术。他们早期的 UI 混淆不堪，有很多地方都是我给他们改了之后，才好了一点。 但是因为我一直不在乎 Python 这语言， 也没觉得这种工具有什么市场， 所以一直没有动手开发一套完整的服务。不是不能做，而是没有动力去做。

BSD 的版权使得 Sourcegraph 的两个创始人可以完全免费， 无限制的使用 PySonar. 这样的结果， 使得我无法为 PySonar 收到任何的回报。后来 Sourcegraph 的两个人找到我，想招我进去，帮助他们制造 RubySonar 和改进 PySonar. 这样就开始了我们的不平等合作关系。Sourcegraph 使用了 PySonar，按理我不需要另外做什么，就应该有一定的回报。然而现在他们把我招进去作为员工，我必须要做点其它事情，才能得到回报， 也就是说我反倒成为了他们的打工仔。几个月之后，我逐渐发现这两个人的肤浅和不尊重。最后，在得到了最重要的技术改进之后，两个创始人翻脸不认人，把我赶出了公司。

新的 PySonar2 里面已经没有了 Google 的代码。由于对人心的失望，我曾一度把 PySonar2 的版权改为 AGPL. 这是 GPL 的增强版，它要求任何使用这些代码的人和公司，在对它做出改进之后，必须把改进的代码能让人下载。就算你在自己的服务器上运行这些代码，不把它作为产品提供给人，也一样需要让人能够下载到改进的代码。这并不是说你不能用这些代码，但如果作为一个公司，你不想让别人得到改进后的代码，那么你完全可以找代码的原作者，付给他一定的报酬，得到闭源使用这些代码的权利。也就是说，AGPL 能够让代码的作者在某些时候得到应有的回报。

把版权改为 AGPL 之后，出现了一个奇怪的事情。申请美国绿卡的时候，我找以前 Google 的上司要一封 “工作经历证明” 。这种证明本应是公司无条件提供的，甚至不应该需要通过上司去获取，而是人事部无条件签发。然而收到 email 之后，前上司却对我说：“我可以给你这个证明，然而我想让你给我提供一个好处作为交换。你的 PySonar2 现在改成了 AGPL 版权，我们想用你的代码，却因为这个版权没法用。你能否把版权改为 BSD 一类的，这样很多人都可以用它？” 面对这样的无理要求，我很鄙视，所以干脆没要 Google 的工作经历证明，直接找其它公司开了证明。

又过了一段时间，我感觉 AGPL 似乎确实限制了 PySonar 的应用，所以又把版权给换成了 BSD, 进而换成了 Apache, 一种比 BSD 还要宽松的版权。刚换成 BSD, 我就发现有一家代码工具公司 fork 了 PySonar, 最新的 commit 正好是改为 BSD 版权的时候。这个公司从来没联系过我，从来没感谢我，只是无声无息地用 PySonar 来赚钱。他们 fork 这个时候的 PySonar, 应该只是用于法律保护，证明 PySonar 的代码在这时候是 BSD 版权的。

这样的作法不仅让我一阵心酸。曾经一直在用 PySonar 的另一家公司的创始人 J, 当天也发信来跟我说：“正在考虑给你版税呢， 结果你就换成 BSD 了。哈哈哈！” 这是什么意思呢？本来都要付钱给你了，结果你把版权换成了 BSD, 所以我就可以不给钱了，捡了个大便宜，就是这个意思吧。他最后还是象征性的给了一千美元，然而这相对于 PySonar 为他创造的价值，其实什么都不是。

我一直把 J 作为朋友。平时如果他报告 PySonar 的 bug, 我乐意免费给他改进。我给他介绍投资人，甚至给他介绍妹子..... 我并没有图他什么，并没有要求回报。我只是想积点德，将来总有好的后果吧。然而，前几天当我宣布离开美国的时候，我才发现我的好心其实并没有好报。J 发信息来，说看我想回国，所以想招我进他的公司。给我开了一个价，具体的数字我就不说了，不过这个工资，现在国内是个程序员都能拿到。我

当时想，给这点钱，给他当个顾问，关键时刻给点方向，隔几个月改改 PySonar 的代码也就算了，结果他说要求我全职给他工作。笑他太抠门，结果得到的回答是嘲笑：“你嫌钱少，可是你做出过什么真正的产品吗？” 哦，是的，PySonar 不是真正的产品。你在外面做个包装界面，然后功劳都是你的了，我还得给你打工？换成 BSD 版权的时候就捡了一个大便宜，现在我要回国，以为我落难了，趁火打劫想捡个超大便宜。说不定到时候还跟 Sourcegraph 一样跟我翻脸不认人..... 哎，我再也不相信任何免费用我代码的人了。

这就是我用 BSD 版权发行有价值的代码的辛酸史。有些人免费拿你的高价值代码去赚了钱，到后来却让你给他做廉价劳工。由于这个原因，从今天起，我的代码完全进入闭源状态。没有人再能免费得到 PySonar 最新的改进，没有人再能看见我最新的技术。另外我还留了一手，PySonar 其实一直以来都有一个未开源的分支，里面含有对于静态分析逻辑的重大改进，能够精确地发现很多 Python 程序里的 bug。从今以后，这一切都属于我私有，它们将会让我未来的产品立于不败之地。

后记：有人可能误以为我要一心拿 PySonar 来做产品。其实 PySonar 支持的那种产品只能用于企业内部提高生产力用，不能产生直接的效益。由于 PySonar 等技术的复杂度和难度之高，收益却不成正比，所以我并不把它作为一个很好的创业手段。我首要的创业目标，应该是创造直接的生产力和效益，而不是提供间接的辅助设施。提供给程序员的产品，程序语言等等，虽然也很重要，然而优先级在我这里其实不是最高的。

5 美国社会的信息不平等现象

在美国工作过的人都知道，进入一个公司之前，雇员都要经过一种 “背景调查” (background check). 这种调查一般由专门的“背景调查公司”来协助进行，他们可以通过各种渠道来获取你的信息，包括身份，住址，犯罪记录，学位信息，之前雇主信息，职位，工资，工作时间，离职原因等等。很多公司还要求你提供几个 “联系人” (reference) 和他们的联系方式，有些甚至要求其中有一个是你之前的 manager，这样他们可以去询问你的表现.....

在美国大学读研究生，进去之前都需要找几个认识的教授写推荐信。进去之后每隔一段时间，教授们会召开一种八卦会议，讨论各个学生的表现。你之前对任何一个教授说的话，都可能传到别的教授耳朵里。教授们用这种机制来打探学生的底细，所以如果你在一个教授那里表现不好（当然其实可能是教授的人品问题）或者发生了矛盾，去找另一个教授的时候很可能立即吃闭门羹，或者找借口回避。在这种会议上，教授们还会决定哪些学生会被 “请离学校”.....

这表面上看上去是为了防止有问题或者不合格的人进入公司或者学校，久而久之你才发现，这种“背景调查”并不是什么好东西。它造成的“信息不平等”，导致了雇员和学生在自身权益保护上处于劣势，陷入被控制，被压迫的地位。雇员和学生如果有问题，公司和学校使用联盟的力量来解决；可是如果公司和教授有问题，学生们却没有相应的机制来维护自己的权益。

所以你经常发现教授欺负学生的情况，最后反而是学生被迫离开。如果一个教授人品有问题导致了矛盾，他总是会推到学生头上。学生只能默默的忍着，绝不会有另一个学生或者教授来维护你的权益，伸张正义。正如中国的一句古话，官官相护，教授和教授之间都是互相庇护的。在美国，教授和研究生是两个地位完全不同的阶级。不要以为在美国你可以对教授直呼其名，在地位上你们就是平等的，那些都是美国一直以来的广告宣传（包括电影，电视，GRE 文章.....）在你头脑里产生的幻觉。

你发现没有，公司和学校可以调查你之前的表现和不良记录，你却没有可靠的办法来调查公司的内部情况和不良底细。如果没有熟人在公司，你是没法知道公司内部的一些龌龊做法的。某些公司里面的情况是多么龌龊，从我之前的文章你应该已经有所了解。你不但不容易找到说真话的"内线"，而且当你进入公司之后会被要求签署一种叫 NDA 的东西，也就是 Non-Disclosure Agreement. 这种 NDA 很多不但要求你不能暴露公司的商业和技术机密，而且要求你不能公开公司内部的 "做法" (practice). 当然做法就包括了内部的各种压榨，政治斗争，勾心斗角，领导瞎指挥，等等。

签了这样 NDA 的人，除非你跟他是很好的朋友，他才有可能在不被抓到证据（无记录）的情况下，亲口告诉你公司内部的真实情况。很多时候，情况要比你从外面看起来糟很多，就算它是世界知名的"伟大"公司也一样。本来你认识公司内部员工的机会就不是很多，再除去本来就精通政治斗争的人，那些喜欢晒幸福显牛逼的人，那些由于签了 NDA 而三缄其口的人，就没有很多机会听到另外方面的信息。

你可以从 Glassdoor 之类的网站了解一些公司的负面信息，然而经验告诉我，Glassdoor 并不是没有 "审查" 的。你大体上说一下不好的感觉可以，然而如果你说到具体的地方，review 就会被 Glassdoor 封锁，理由是里面有脏话，或者违反 "社区规则" (community guidelines). 你以为真是因为脏话吗？等你删掉所有的脏话和用星号替换的脏话（比如 s**t），会发现仍然无法通过审查。他们不会告诉你为什么，只是反复的跟你说违反了社区规则。至于怎么违反了，你是永远琢磨不出来的。

到底哪里有问题呢？问题就在于你的 review 太具体了，包含了确凿的证据，别人一看就知道那是真的。Glassdoor 的所谓社区规则，让你无法显示出具体的证据，以至于人们看到你的反面 review 也无法确信你说的是实话。有些甚至可能以为你是个人性格问题，要求太高，对公司不满而已。当你忽略了这些反面 review, 进入公司一看，才发现他们说的都是真的..... 所以像 Glassdoor 这种试图朝 LinkedIn 竞争者方向发展，目的在于盈利的公司，它们其实是不敢让这样的 review 存在的。否则得罪了某些牛气的公司或者投资者，自己可就吃不了兜着了。要记住 Glassdoor 也是一家公司，然而能够给公司提供公正 review 的地方，它自己绝对不应该是一家公司。

由于新的工作都需要背景调查，甚至要求给你之前的上司打电话，所以知道这一点的人都会在工作中唯唯诺诺，不敢得罪任何人，就算你的上司人品非常差也一样。这样一来，背景调查和推荐制度，就成为了管理层控制工薪阶层和学生强有力的工具。你的上司比较放心，无论他如何瞎指挥，如何欺负陷害你，你都得在他面前笑嘻嘻的，不敢当面冒犯。如果他对你不满意，就算你离开这个公司，将来的工作也不好找。因为之后的公司可能要求打电话跟他询问你的情况，到时候他可以在背地里狠狠黑你一番，然后还跟你说自己高度的赞扬了你。等你过五关斩六将，到了最后却莫名其妙没有拿到以为可以顺手拈来的工作，才发现他也许是一只笑面虎。

我在 Google 和 Coverity 的两个上司都是这样的人。其中 Coverity 的上司曾经在 2013 年直接导致我失去一个很好的工作机会，幸好本来要给我 offer 的公司里一个好心人事后告诉了我原因 —— Coverity 的 manager 提供了"让人震惊的负面信息"！在那之前我一直以为，虽然这人帮着公司压榨我们，但平时又跟我哭诉说是被迫的，所以也许还有点人性。当新公司要求之前公司的联系人中包含一个 manager 的时候，由于只有 Google 的那家伙和他可以选，我写了他的名字。结果呢，同事给了我好评，然后这个 manager 本来的面目就显示出来了..... 现在你知道美国的人际关系可以多么的微妙和复杂了吧，一个人在背后戳你一刀，你也许直到最后都没有发现，而且很多时候你的新公司明确要求你提供让别人捅你一刀的机会.....

这就是中国人心目中简单纯朴善良的美国人，他们制造了世界上最强大的无形锁链，根本不需要政府出手，利用整个社会的集体力量来操控和挟制每个人的行为，使得他们不敢有不服从的举动，不敢公开公司和学校的不良做法。我之所以在博文里曝光一些公司的行为，就是为了抵抗这种信息的不平等，为了破坏这种无形的锁链。

有些人每次看到我批评前任雇主，就觉得我是大嘴巴，说得好像我在欺负雇主一样。我这种资产为负数的区区小民（负人阶级），哪里能动得了牛逼哄哄的 Google, 不可一世的 Coverity 的汗毛呢？可是你发现没有，被我曝光的全都是强权势力，而且它们都对我的身心健康和切身利益产生了严重的危害。我从来没有说过我前女友，我前同事的故事，就算他们有些做法相当的不好，也绝对不会公开出来。因为他们没有像公司那样强大的权力和危害，可以伤害大量的人却不受惩罚，可以导致整个社会文明的败坏。所以就算我受到很大的伤害，也要保护这些人的隐私，因为他们还有醒悟和改进自己的机会。

然而对于公司和大学这样的强权，如果出现严重侵害切身利益的恶劣行为，我就会毫不留情的揭露。就算 NDA 禁止透露某些信息也一样，我不会让 NDA 阻碍我对伤害自己的恶劣行径进行披露。就像你被别人打了，打人者逼你签字画押，让你保证不说出去，不然上法院告你。你去遵循这种条款不是很可笑吗？本来该被告上法庭的是打人者，到头来打人者却要挟上法院告你，试图封你的嘴，掩盖真相。写 GRE 作文的时候大家都分析过如何对待“unjust law”（不正义的法律），现在是拿出来用的时候了，用于掩盖公司丑恶行径的 NDA 条款是不正义的法律，所以我们应该联合起来废除它！当然这里说要揭露的，不包括商业和技术机密。

这里我应该强调一下，并不是所有问题都属于“严重侵害”，都需要进行揭露。这些侵害只包括那种对人的身心健康，切身的基本利益，也就是所谓“人权”产生危害的。比如利用高压甚至威胁剥夺员工休息时间，危害员工生命安全和健康，暴力谩骂等行为，缺乏对人的基本尊重的行为，才列入被揭露之列。其它的常见问题，比如节奏太慢，工作缺乏挑战性，process 太繁琐，同事比较笨，..... 由于没有造成伤害，所以不包括在内。

一些典型的公司恶劣行径例子：

1. Coverity 故意对任务设置过短的时间，然后通过解雇相威胁，导致员工严重超时工作，无耻剥夺员工的休息时间。
2. Sourcegraph 在员工短时间完成重大贡献之后，使用无理借口解雇，收回早期员工的大额股票份额。这是相当于抢劫的犯罪行为。
3. Amazon 被多次曝光的极度压榨的工作环境，对怀孕女员工的不公正待遇，等等 .....

有人跟我说我这么曝光以前的雇主，新的公司会对我有所顾忌，这样的行为等于自杀。我其实一点都不担心这个事情。虽然恶劣的行径是一定会被揭穿的，然而胸怀坦荡，对人友好，心里没有鬼的公司却大可不必担心。对我有所顾忌是应该的，我理所应当有自己的威严——公司都应该知道，王垠不是好惹的，但他却是非常讲理的。你可以看到，被曝光的雇主都有非常严重的恶劣行为，甚至可以告上法庭，要求赔偿。我对这样的公司躲都来不及，如果类似的不地道的公司看到了我的文章而回避我，那正好！因为我正想很有效的过滤掉这样的公司，省得浪费时间去跟他们聊。剩下的有良心的公司，自然会发现我是朋友，而且是非常有价值的人，从而愿意跟我合作。

如果你被公司欺负了，却担心曝光了公司的恶劣行为会导致以后找不到工作，那你就助长了这种公司的气焰。他们就会打着假面具继续害人，把大家都蒙在鼓里。这样坏人就会横行霸道，导致整个社会环境的恶化。所以这些人对我的做法的担心，说我是在自杀，不但是多余的，而且是有害的，甚至可以被视为一种恐吓行为。我希望广大的劳动群众都能有如此的勇气，勇于站出来说真话，世界才能得到信息的平等。只有在信息上平等，公司的不良行为才能受到节制，有良心的公司才能得以发扬光大，人类才有可能得到物质上平等的机会，最终消灭人压迫人，人剥削人的制度。

6 美国企业的装嫩问题

知道从什么时候开始， 美国的大小公司， 都开始重视所谓 “企业文化”， 仿佛企业一定要有自己独特的文化，不然就不够 “酷”，就不能吸引人了。没见过世面的大学毕业生， 很容易因为某些公司鼓吹的“年轻文化”而加入他们， 进去之后才发现不爽。 作为一个经历过这一切的人，我觉得有必要把这个问题拿出来专门说一下。

说到喜欢鼓吹自己企业文化的公司，首先想起来的当然是 Google 了。Google 总是号称 “不作恶” (don’t do evil), 声称自己虽然是大公司，却仍然保留了 startup 的文化。进入 Google 时，HR 都会告诉你要 "Googley", 要接受 Google 的价值观，融入 Google 的文化。 在 Google， 到处是鲜亮的 "Google 色" 设计， 很多人的办公桌上摆放着稀奇古怪的魔方， 各种谜题和玩具， 显示自己很聪明或者有创造力。办公室各个角落都有 foosball，台球桌，游戏机，Rock Band 一类的游乐设施。时不时还有人发动 NERF 枪战，一颗颗的泡沫子弹从你头顶上呼啸而过..... 初进 Google，你也许会感叹，这是一个多么年轻的公司，简直跟游乐场一样！

确实，Google 的文化属于 startup 文化，然而 startup 文化真的好吗？我在好几个真正的 startup 待过，所以我知道他们确实也是那个样子。很多 startup 的办公室采用完全的 “开放空间” 设计，办公桌之间没有隔离板，同事之间除了某些角度可以用显示器遮一下，基本没有隐私可言。某些 startup 甚至把电脑摆成一排排的，让程序员肩并肩地坐在一起工作，跟中学生上课的教室一样。当然，各种玩具，游戏机，NERF 枪，都是不可少的。很多 startup 在管理上还采用 Agile 方法，每天早上开站会，Scrum，在全公司采用结对编程 (pair programming), 引入各种新的程序语言 (Scala, Clojure, Go, Haskell) 和其它“新技术”，经常自己一拍脑门想出与众不同的团队合作方式，就立马开始在全公司推行..... 这些公司很想制造一种 “融洽”，“合作”，“年轻”，“创新” 的氛围，然而经验告诉我，这样的做法几乎总是得到适得其反的效果。

这些“新奇”，“年轻”的企业文化，似乎很容易吸引刚走出学校，未经世事的年轻人，然而在一个成熟的人看来，这些公司并没有显示出真正的活力，而是显示出愚蠢和虚伪。事实证明， 缺乏隔离措施的所谓“开放空间”办公室， 并不如传统的单间办公室或者隔离间（cubicle）。编程需要的是独立思考，思考需要集中注意力，很多人思考时甚至不喜欢别人能直接看见自己。 没有隔离和隐私措施的办公室， 使得员工之间随时都可以互相干扰， 难以集中注意力， 这对于写出优雅清晰的代码是很不利的。 要是遇到那种成天高谈阔论的人在你旁边，那就更倒霉了， 不得不成天戴着耳机工作。 有些人时不时地发动 NERF 枪战， 不但极大地干扰其他人的正常工作， 而且制造一种浮躁虚假的氛围， 你甚至可能因此受伤。至于 Agile，站会，试用新语言，采用结对编程，鼓吹 TDD，突发奇想的新管理方式， 等等， 都是 startup 的通病。 很多新公司没有经验， 不知道什么是真正好的技术和管理方式， 再加上不小心招进来不懂装懂的管理者， 所以走一步错一步，浪费大量的资源，一步步地走向失败。

在我眼里，Google 就像一个上了年纪的大妈，偏偏要染了彩色头发，穿上日本少女装，嗲声嗲气地说话，各种卖萌，让人起鸡皮疙瘩。这是什么问题呢？这里有一个 “得体”的问题。一个人的穿着和行为，必须符合他的实际年龄或者更加成熟，才会显得得体和优雅。 一个小男孩完全可以穿上笔挺的西装， 小大人般的稳重也不会招人厌恶。 然而一个大叔或者大妈，穿上少男少女装，嗲声嗲气说话，那就会让人恶心。事实是， 不管人的年龄如何， 成熟稳重优雅才是正道。 年龄和经验应该被作为一种值得尊敬的财富， 而不是一种需要隐藏的弱点。 我们可以容忍一个人在年轻的时候略显幼稚和浮躁， 却难以接受有一定生活经历的人显示出这样的气质，因为我们期望他们已经学会一些东西。

当然，我并不是说 startup 采用这样的文化就可以接受。其实不管公司成立时间有多短，采用这种 “装嫩文化” ，都是一样的问题。公司成立的时间虽然短，然而公司里的人却都已经是大学毕业的成年人， 所以就算再“年轻”的公司， 把公司文化搞得跟中学似的， 只会让有经验的人笑话。 每一次进入这种刻意制造年轻氛围的公司， 或者跟他们面试， 我都会发现这公司存在非常奇葩和愚蠢的问题， 所以我后来都避免接触这种大肆鼓吹自己文化的公司。

美国公司这些所谓 “企业文化”， 都是刻意做出来的表面现象， 换句话说就是 “装”，或者叫 “套路”。无论表面上多么 “和谐” 和 “年轻”，都是假的。 公司装嫩的目的， 往往在于掩盖他们本质上的问题。把自己的 “文化” 强加于人，说明他们在本质上是不尊重人的。

一个好的企业文化， 应该是每一个员工自然而然从自己的生活中带来的， 而不应该是公司创始人独断专行 “制造” 出来的。一个好的企业，首先应该对每一个人发自内心的尊重和真正的关心，给他们提供良好的，可以安心工作的环境，提供好的福利，剩下的所谓 “文化”，不需要你刻意去制造，自然而然就有了。

据我了解， 这种装嫩现象是美国公司特有的，欧洲和中国的大部分公司都没有这种问题。 不过我也知道， 由于很多中国人对美国的崇拜心理，再加上一些从这类美国公司海归的中国人，有些国内的 startup 也开始采用类似的装嫩作法。在这里我想告诉广大的中国企业，我们其实没必要模仿美国公司的这种文化，它并不是什么好东西。 我也想告诫广大求职者，见到这种鼓吹 “年轻文化” 的公司，最好提防着点 :)

7 我不是编译器专家

工作多年以来，我深刻体会到一个现象，那就是做过 “编译器” 工作的人，哪怕只做了点皮毛，都容易产生高人一等的心理， 以至于在与人合作中出现各种问题。 由于他们往往也存在偏执心理和理想主义， 所以在恶化人际关系的同时， 也可能设计出非常不合理的软件构架，浪费大量的人力物力。

我曾经提到的 DSL 例子，就是这样的两个人。他们都自称做过编译器，成天在我面前高谈阔论，甚至在最基础的概念上班门弄斧，显示出一副 “教育” 其他人的姿态。其实他们只有一个人做过 parser, 还不算是真正的编译器工作，却总显示出高深莫测的模样。哲人一样捋捋胡子，摇摇脑袋，慢条斯理，嗯..... 另外一个完全就是外行，只是知道一些术语，成天挂在嘴边。每次他一开口，我都发现这个人并不知道他自己在说什么，却仍然洋洋得意的样子。

我是被他们作为专家请来这个公司的， 来了之后却发现他们最喜欢的事情， 是在我面前显示他们才是 “专家”。他们也问过我问题， 可是每一次我都发现他们并不想知道答案， 因为我说话的时候他们并没有在听。不管说什么问什么， 他们似乎只想别人觉得他们是最聪明的人。

虽然对其他人趾高气昂，全懂了的样子，对于 Brendan Eich（JavaScript 语言的创造者）这样有权势的人物，却是各种跪舔，显示出各种 “贱” 来。我虽然尊重 Brendan Eich 个人和他的语言，然而很明显他是半路出家，对语言设计并没有很深的造诣。对语言稍微有点研究的人，都不会对这种人物显示出谄媚的态度。

"Yin, 你知道 X 吗？" 当然他期望的是你说不知道，这样他就能像大师一样，把这个刚学到的术语给你讲半天。每当这个时候，我就想起一个前同事喜欢说的一句话：“你问我，是因为你不知道，还是因为你知道？” 其实他问的这个概念 X, 常常是我很多年前热心过，试验过，到最后发现严重问题，抛弃了的概念。

更糟的事情是，这其中一人还是 Haskell 语言的忠实粉丝，他总是有这样的雄心壮志，要用"纯函数式编程"改写全公司的代码……

遇到这样的人是非常闹心的，到了什么程度？他们经常雄心勃勃用一种新的语言（Scala，Go 之类）试图改写全公司的代码，一个月之后开始唾骂这语言，两个月之后他们的项目不了了之，代码也不知道哪里去了。然后换一种语言，如此反复……

后来实在没做出什么有用的东西，这两个人又突发奇想，开始做 DSL, 闹得团队不得安宁，有点资历的工程师（包括我和一位早期 Netscape 的资深工程师）都极力反对，向大家指出更容易，更省力的解决方案。然而由于管理层根本不懂，所以任凭这两个人拍胸脯，没有困难制造困难也要上。因为烦于他们在我面前高谈阔论，而且对这个 DSL 的事情实在看不下去了，我干脆换了一个部门，不再做跟语言和编译器相关的事情。

现在这个 DSL 做了好几年了，仍然很垃圾，然而公司人傻钱多，居然请到了 Java 界的资深人物来给这 DSL 写 specification. 这两人也分别升职为 Principal Engineer 和 Distinguished Engineer. 当看到 "Distinguished Engineer" 这个 title, 我觉得太好笑了。当然，我相信有资历的 PL 人都会明白这 DSL 的问题，我想象着这位 Java 人跟这两人将会发生的冲突。如果他对此没意见的话，那他的水平还真是值得怀疑了。

在 Coverity 和其它公司遇到的编译器人，基本是差不多的问题。他们下意识里把自己看成是最高档次的程序员，所以对其他人显示高高在上的气势。

Coverity 有一个 ABC 工程师，因为自己写过完整一点的静态分析，比较会折腾 C++, 总是趾高气昂的对待其他人，甚至直接对别人说："你写的这是什么代码啊？我绝对不会写出这么烂的代码！" 还有一个从斯坦福编译器教授 Alex Aiken 那里毕业的 PhD, 在 Coverity 做构架师，平时一行代码不写，也不看其他人写的，说不出见解深刻点的话，因为与实际工程脱节，尽在瞎指挥。地位最高的 Distinguished Engineer, 成天优哉游哉，看一些关于 parser 的话题，似乎 parser 是他终身的研究方向，也不做什么实事。

我所在的每一家公司，只要工作跟编译器沾边，总是不免遇到这样的人。其它的我就不细讲了。

有些美国公司在招人的时候表示，对简历里提到 "做过编译器" 的求职者有戒备心理，甚至直接说 "我们不招编译器专业的人"。以至于我也曾经被过滤掉，因为我做过编译器相关工作。编译器专业的人本来可以做普通的程序员工作，为什么有公司如此明确不要他们呢？我现在明白为什么了，因为自认为是 "编译器专业" 的人，有大概率是性格很差的团队合作者，喜欢显示出高高在上，拯救世界的姿态，无法平等而尊重的对待其他人。

有些人也把我叫做 "编译器专家"，喜欢在我面前提 "编译器" 这个词。我一直听着别扭，却没有正式拒绝这个称呼。每每遇到 "真正" 的编译器专家，我总觉得自己不是那个圈子的。不是我不能做编译器的工作，而是编译器领域人士的认识水平，理念和态度和我格格不入。

所以我应该明确表个态：我不是编译器专家，而且我看不起编译器这个领域。我一般不会居高临下看低其它人，然而对于认识肤浅却又自视很高的人，我确实会表示出藐视的态度。现在我的态度是针对编译器这整个领域。真的，我看这些人不顺眼很多年了。

就最后研究的领域，我是一个编程语言（PL）研究者，从更广的角度来看，我是一个计算机科学家。有人听了 "科学家" 一词总是误以为我在抬高自己，而在我心目中 "科学家" 仅仅是一个职业，就像 "厨师" 一样，并不说明一个人的水平和地位。PL 研究者被叫做 "计算机科学家" 是很恰当的，因为 PL 领域研究的其实不只是语言，而是计算的本质。通常人公认的计算机科学鼻祖 Alan Turing 也可以算是一个 PL 研究者，虽然他认识水平比较一般。

IT 业人士经常混淆编程语言（PL）和编译器两个领域，而其实 PL 和编译器是很不一样的。真懂 PL 的人去做编译器也会比较顺手，而编译器专业的却不一定懂 PL. 为什么呢？因为 PL 研究涵盖了计算最本质的原理，它不但能解释语言的语义，而且能解释处理器的构架和工作原理。当然它也能解释编译器是怎么回事，因为编译器只不过是把一种语言的语义，利用另外一种语言表达出来，也就是翻译一下。PL 研究所用的编程范式和技巧，很多可以用到编译器的构造中去，但却比编译器的范畴广阔很多。

深入研究过 PL 的人，能从本质上看明白编译器里在做什么。所以编译器算是 PL 思想的一种应用，然而 PL 的应用却远远不止做编译器。每次有人说我是做编译器的，我都觉得是一种贬低。我只不过拿精髓的理念稍作转换和适应，做了点编译器的事情，就被人叫做"编译器专家"，而我根本不是局限在这个方向。

专门做编译器的人，一般是专注于"实现"别人已经设计好的语言，比如 C，C++. 他们必须按照语言设计者写好的语言规范（specification）来写编译器，所以在语言方面并没有发挥的空间，没有机会去理解语言设计的微妙之处。

许多做编译器的人并不是从零开始写的，而是拿现成的编译器来修改，所以他们往往被已经存在的，具体的构架限制了想象力。极少有编译器人完整实现过一个语言，都是在已有的基础上小改一下，优化一些局部的操作。这大大限制了他们可以获得的全局洞察力。

很多编译器工程师并没有接受过系统的 PL 理论教育，有些甚至是半路出家，在学校里根本没碰过编译器，也没研究过 PL。比如我的第一个公司 Coverity，招进去的很多人从来没碰过编译器，也不懂 PL。我进去不久，Coverity 的 VP 满口牛气向新人宣布："我们会教会你们一切！" 然而很可惜，PL 的精华根本不是一个公司在短期能够传授的。Coverity 没有这个能力，Google, Facebook, Intel, 微软..... 都没有这个能力。

很多半路出家的编译器工作者以为在公司跟着做项目，折腾下 LLVM 之类，就会明白所有的原理。然而事实是很多人这样做了十几年，仍然不明白最基础的原理，因为他们被具体的实现限制了想象力。PL 理论联系着计算的本质，不明白这些原理就只能看到肤浅的表面，死记硬背，遇到新的现象就没法理解了。跟 LLVM 专家聊天，我很多时候发现他们的知识是死的，僵化在 LLVM 具体的实现里了。

由于缺乏对 PL 理论的深入研究，编译器人往往用井底之蛙的眼光来看待语言，总以为他们实现过的语言（比如 C++）就是一切。一个语言为什么那样设计？不知道。它还可以如何改进？不知道。"它就是那个样子！" 这是我常听编译器人说的话。当然很多编译器人连 C++ 都没法完整实现，只是在已有基础上做了很小的改动。

许多编译器人把 C++ 的创造者 Bjarne Stroustrup 奉为神圣，却不知道 Stroustrup 在 PL 领域并不是闪耀的明星。Stroustrup 曾经在 2011 年 11 月 11 日来到 IU 进行关于 C++11 的演讲，IU 的资深 PL 教授们都有到场。Stroustrup 谦卑的说："我需要向你们学习很多东西来改进 C++。" 看似 "谦虚"，其实他说的是实话，因为 IU 的教授们在语言设计上确实比他强很多。

Stroustrup 的整场演讲，我没有看到任何新颖的突破，全都是几十年早已出现，我天天都在用的东西。然而这些 C++ 的改进被编译器人看作是重大的历史性的突破，因为他们很多人根本没用过其它语言，甚至不知道它们的存在。

后来我的一个能力比较弱的 PL 同学进入了 C++ 委员会，为改进 C++ 做一些事情。从她的描述和表现，我感觉 C++ 委员会气氛十分的官僚，古板和愚钝。她进了 C++ 委员会之后，感觉整个人都傻了一样，很肤浅的小事也说得眉飞色舞，好像什么重大的突破一样。真懂 PL 的一些同学，很少有混进 C++ 委员会的，因为那意味着要利用另外的关系网，让一些自己根本看不起的人骑在自己头上，必须先帮他们做一些瞎扯淡的事情。

编译器人所膜拜的大师，在真正的 PL 研究者眼里其实不算什么。编译器人与 PL 研究者在见识上的差距是非常明显的。PL 人因为看透了很多东西，比较谦虚，往往不想揭穿编译器人的差距。但编译器人却因为在“工业界”有地位，趾高气昂以为自己懂了一切一样，结果遇到深刻点的 PL 问题就各种稀里糊涂。

实际上做编译器是很无聊的工作，大部分时候只是把别人设计的语言，翻译成另外的人设计的硬件指令。所以编译器领域处于编程语言（PL）和计算机体系构架（computer architecture）两个领域的夹缝中，上面的语言不能改，下面的指令也不能改，并没有很大的创造空间。

编译器领域几十年来翻来覆去都是那几个编程模式和技巧，玩来玩去也真够无聊的。起初觉得新鲜，熟悉了之后也就那个样了。很多程序员都懂得避免“低水平重复”，可是由于没有系统的学习过编译器，他们往往误以为做编译器是更高级，更有趣的工作，而其实编译器领域是更加容易出现低水平重复的地方，因为它的创造空间非常有限。

同样的编译优化技巧，在 A 公司拿来做 A 语言的编译器，到了 B 公司拿来做 B 语言的编译器..... 大同小异，如此反复。运气好点，你可能遇到 C, C++, Java. 运气不好，你可能遇到 JavaScript, PHP, Ruby, Go 之类的怪胎，甚至某种垃圾 DSL. 但公司有要求，无论语言设计如何蹩脚，硬件指令设计如何繁琐，你编译出来的指令必须能正确运行所有这语言写出来的代码。你说这活是不是很苦逼？

我在 Cornell 的时候，有一个很有权势的编译器教授，从未发表有理论价值的 paper, 却老在 Java 上面做文章。他和他的博士生们总是把一些其它语言几十年前已经有的 “新特性” 搬到 Java 上面，老酒换新瓶，发 paper 拉 funding. 由于拉了很多钱，所以在系里很受宠，他的学生们在其它人面前都趾高气昂的样子。

后来这教授的一个学生去了 Facebook, 帮他们做 HipHop, 一个从 PHP 到 C++ 的 “编译器”。其实这种 “源到源” 编译器做起来不算难，但给 PHP 这样劣质的语言做编译器，实在是狗血的工作，繁琐而头痛。没有任何理论价值不说，在工业界有什么价值也难说。我的一个前同事曾经对 Facebook 的这个项目发表了一个尖锐而幽默的评价：“Facebook 现在不但给母猪涂上了口红，而且真的开始 f.. 它了！”

后继的还有 PHP VM 一类的东西，越来越离谱。后来这位同学可能也受不了，换组去做其它跟语言无关的事情了。在 PL 研究者看来，VM 也并没有什么稀奇。PL 领域有各种各样的 “抽象机”（abstract machine），比如 CEK machine，它们揭示了计算的方方面面。我自己都设计实现过好几个 “可逆抽象机”，它们可以进行所谓 “可逆计算”。所以一个 PL 研究者很容易就能设计出一个 VM 来，它们只不过是一种经过部分优化的解释器。

每每看到编译器人说到 “VM” 这个词的时候那种荣耀而敬畏的神情，好像只有他们明白 VM 是什么，我就觉得好笑，外加一种说不出的滋味。编译器人虽然知道一个具体的 VM 怎么实现，知道一些死板的细节和术语，却不知道 VM 的本质是什么，不知道一个全新的，具有新特性的 VM 要怎么设计出来。

在《Chez Scheme 的传说》一文中，我提到在 Cornell 的时候选过一门编译器课程，后来在半学期的时候 drop 掉了。现在回想起这段历史，发现它对 “教育理念” 这件事挺有启发意义。教育是什么，是为了什么？Cornell 的这门课给了我一个很好的反面教材。

这个编译器课程那一年的教授是 Tim Teitelbaum，他也是 GrammaTech 公司的创始人。GrammaTech 是与 Coverity 类似的静态分析工具，不过 GrammaTech 还能分析二进制代码。Tim Teitelbaum 是 Donald Knuth 的崇拜者，他经常提到 Knuth 提出的一些“伟大概念”，比如 attribute grammar. 总是把 Knuth 那些东西说成是最伟大的发明。

这门课不知道最初是谁设计的。Andrew Myers 和 Tim Teitelbaum 以前交替着讲这个课。

那么我为什么会 drop 这门课，而且是在学校允许 drop 课程的 deadline 之后呢？因为它的教育理念非常的落后和不合理，可以说就是坑人的。

从课程的大纲你可以看出来，它是很传统的编译器课程，一开头花很多时间精力去折腾 parser. 源语言是一种类似 Java 的语言，parser 是使用类似 lex, yacc 的工具生成的。这种盲目重视 parser 的误区，我已经在另外一篇文章批评过，但还这不是我鄙视的重点。

这门课最让人受不了的事情，发生在我成功完成 parser, 开始编译代码的第一个 pass 之后。当得到那次作业分数的时候，我惊呆了。我从来没有得过这么差的分数！仔细看原因，说我的代码没通过好些“测试”。我到那个时候才明白，原来提交后的代码，会被助教拿来跑一些我毫不知情的测试（test），然后他简单的根据这些测试的结果给出分数。

作业本身的要求是用大段大段的英语写下来的。你需要按照这些英语描述从零实现编译器。真的是从零开始，没有任何的框架或者示例代码，完全从白纸开始。经过许多努力，你写出了编译器，还自己写了一些小测试，你觉得完全满足了作业的要求。可是提交之后，你的编译器代码却要被一整套你手里没有的“测试”进行检验。所以最后你惊讶的发现，自己以为做对了，而助教那里的测试有那么多没通过！

最让人无语的事情是，学生手里是没有这套测试的，而且他们不给你。也就是说，你提交作业的时候，无法用最后给你评分用的那些测试来跑你的编译器，所以你无法知道提交之后会有多少测试失败。

当我向助教和教授抗议，说这样不合理，要求得到那些测试的时候，我受到粗暴的拒绝和鄙视。那种语气，好像是在说我是一个不合格的学生，提一些无理要求。用来打分的测试怎么可能给你，你是太笨了吧？

很多其它 Cornell 学生被这样对待，可能都以为没什么，按照他们的要求做就行了，然而这是完全不合理的。按照合理的教学理念，学生应该有权得到自己学习状态的反馈。如果学生做这种编程作业，就应该能从实际的测试中得到反馈，知道自己的编译器是否符合要求。要知道，大段大段的英语描述，是很容易漏看或者误解的。只有大量的测试才能正确的抓住“要求”本身。所以不给测试，就相当于不给你准确的要求，到后来却要拿这套测试来给你打分。

课程本来应该把测试连同英语描述一起给学生，他们实现之后，自己跑通所有测试，再提交代码。这样学生就能准确的把握作业的“要求”，而不是看着那些混淆不堪的英语段落自己在那里猜。

因为这个原因，而且由于教授和助教的傲慢态度。我最终决定在课程都快进行到一半的时候 drop 这门课程。当然，要进行这个操作是需要系主任签字特许的，为此我还在系主任那里留下一笔“污点”。

在我看来，Cornell 教授们的这种做法，根本就不是合格的教育者，可以说就是在坑人，整人，害人。在他们的理念里，教育是单方面的，学生必须通过作业和考试，而教授却不需要为教学方法负责，可以随便怎么教，作业和考试想怎么整都行。

很多 Cornell 教授有类似的现象，教学不用心，光是各种拉 funding，耀武扬威，完全不顾学生死活。也许这就是为什么 Cornell 总是有学生自杀。我走了之后有一年，在一个星期之内有三个学生从学校里瀑布旁边的吊桥跳下去自杀，新闻轰动了全美国。

后来在网上看到有人骂 Cornell，说：“Cornell 想教你游泳，于是他把你推进池塘里，等你扑腾上岸。等你快上来的时候，他又朝你扔一块大石头，然后继续等你游上来。等你又快上岸了，他又拿起一个榔头往你头上猛砸。这样你就可以死了，可是 Cornell 仍然在那里等着你游上岸来.....”

这段话恰到好处的描述了我的在 Cornell 的经历。

转学到 IU 之后，我参加了 Kent Dybvig 的编译器课程，发现我所设想的编译器课程原来早已被他实现了，而且实现的如此友好。编译器的每一个 pass，都会把所有的“官方测试” 发给学生。 学生按照要求实现每个编译器 pass， 在自己电脑上跑通所有测试， 充分检查，然后才提交作业。 而且作业的网站会自动测试你提交的代码， 在提交的当时就给你反馈：“你有 N 个测试没通过，请修改后重新提交。”

这才是正确的教育方法， 因为它给予学生合理的反馈，让他们清晰的知道自己的表现是否符合预期，主动进步， 而不是拿一些学生事先不知道的标准在那里瞎坑人， 光是给人打分。

Cornell 没有明白教育的目的是培养人，而不只是给人发文凭。Dybvig 教授不但技术和学术水平远高于传统的编译器人， 而且他的课程也设计得如此科学和友好。 这才是真正的教育者。

虽然苦逼，编译器人往往自高自大，高估自己在整个 IT 领域里的地位，看低其它程序员。编译器人很多认为自己懂了编程语言的一切，而其实他们只是一知半解。

编译器领域最重要的教材，龙书和虎书，在我看来也有很多一知半解，作者自己都稀里糊涂的内容。而且花了大量篇幅讲 parser 这种看似高深，实则肤浅的话题，浪费读者太多时间，误导他们认为 parser 是至关重要的技术。以至于很多人上完编译器课程，只学会了写 parser, 对真正关键的部分没能理解。龙书很难啃，为什么呢，因为作者自己都不怎么懂。虎书号称改进了龙书，结果还是很难啃，感觉只是换了一个封面而已。

我曾经跟虎书作者 Andrew Appel 的一个门徒合作过， 当时这人在 IU 做助理教授。借着一次我跟她做 independent study 的机会，逼我写毫无意义的论文，而且对人非常的 push 和虚伪。作为普林斯顿大学毕业的 PhD, 学识水平跟 IU 的其他教授格格不入，却在待人接物方面显示出各种 “贱”，对编译器领域的 “牛人” 各种跪舔，随时都在显示自己以前在某某人身边工作过。那是我在 IU 度过的最难受的一个学期，这使我对 “编译器人” 的偏见又加深一层。

编译器领域的顶级人物如此， 其它声称做过编译器的人也可想而知了。 大部分自称做过编译器的人，恐怕连最基本的的编译器都没法从头写出来。利用 LLVM 已有的框架做点小打小闹的优化，就号称自己做过编译器了。 许多编译器人士死啃书本，肤浅的记忆各种术语（比如 SSA），死记硬背具体实现细节（比如 LLVM 的 IR），看不透，无法灵活变通。

所以我常说，编译器是计算机界死知识最多， 教条主义最严重的领域。经常是某人想出一个做法，起个名字，其他人就照做，死记硬背，而且把这名字叫得特别响亮。你要是一时想不起这名字是什么意思， 立马被认为是法国人不知道拿破仑， 中国人不知道毛泽东。你不是做编译器的！

现在因为 AI 的泡沫，很多人转向所谓 “AI 框架”， “AI 编译器”。 这类职位如此之多，以至于很多人根本没碰过编译器，也摇身一变成为了 “深度学习编译器工程师”。

半路出家的 “AI 框架工程师” 和 “AI 编译器工程师” 们，在别人写出来的框架上小打小闹优化一下，就以为自己做的是世界上最前沿的工作，却不知道深入研究过 PL 的人其实很容易就看破了那些东西。很多 AI 框架工程师嘴里各种奇怪的术语，却看不透所谓 “AI 框架” 只不过是 “可求导编程语言”，完全不能从高级语言和逻辑的角度去看问题。

AI 框架和编译器里面的原理和本质很容易被 PL 理论解释，PL 研究者能够为这些项目指出正确的方向，避免不必要的弯路，然而这些自诩为 “编译器人” 的 AI 框架工程师们完全意识不到这一点。 自高自大， 膜拜权威， 完全没有去听 PL 研究者在说什么，甚至觉得能 “教育” 比自己看得透的人。

每一个大公司都要趁着 AI 这个热度做自己的 “AI 框架”，“AI 编译器”，唯恐不做自己的框架，就会在业界丢面子，所以一窝蜂而上。一定要聘用名声很大的 AI 框架专家来公司站台，虽然也不知道他最后能做出什么来。所有 AI 框架和编译器都大同小异，属于无谓的重复劳动。有些人捣鼓一下这个框架，然后用同样的技巧去捣鼓另外一个，中间都是一些工程性的脏活。这种事情真是非常无聊。

AI 的热潮正在褪去，大部分 AI 公司会在一年之内失败。“AI 编译器” 的工作也会濒临灭绝。所以任凭他们自己瞎蒙乱撞吧，反正坚持不了多久了。

这就是为什么虽然有多次编译器的工作机会，包括 Apple 的 LLVM 部门，我最后都没去。进入 Intel 的时候，本来编译器部门也欢迎我，可是再三考虑之后还是选择了其它方向。因为我很清楚的记得，每一次做编译器相关工作都是非常压抑的，需要面对一些沉闷古板而自以为是的人，而且内容真的是重复，无聊和枯燥。

我唯一敬佩的编译器作者是 Kent Dybvig, 但我也不想跟他一起做编译器。最近很多芯片公司的 “AI 编译器” 部门找我，我全都拒绝了。我不喜欢身边围绕着这些人，做着这些事。我宁愿去卖烧饼也不想做编译器。

由于编译器人的性格特征，除非一个公司专门要做编译器，否则对于曾经做过编译器，想换个方向的求职者，在面试的时候最好深刻了解他们的性格，态度和做事方式，看他们是否能看淡这些，能否平等对待其他人，能否理性而实在的对待工程。否则自视很高的 “编译器人” 进了公司，很可能对团队成为一种灾难。

我写这篇文章是为了警醒广大 IT 公司，也是为了在精神上支持其它程序员。我希望他们不要被编译器的 “难度” 迷惑了，不要被编译器人吓唬和打压。你们做的并不是更低级，更无聊的工作。正好相反，真正可以发挥创造力的空间并不在底层的编译器一类的东西，而在更接近应用和现实的地方。

每当有人向我表示编译器高深莫测，向往却又高攀不上，我都会给他打一个比方：做编译器就像做菜刀。你可以做出非常好的菜刀，然而你终究只是一个铁匠。铁匠不知道如何用这菜刀做出五花八门，让人心旷神怡，米其林级别的菜肴，因为那是大厨的工作。要做菜还是要打铁，那是你自己的选择，并没有贵贱之分。

洞見篇

1 我为什么不再做 PL 人

我不做程序语言（PL）的工作已经半年了。在这半年里，我变得快乐了很多，对世界也有了新的观点。现在我想来讲一讲，我为什么不想再做 PL 的工作和研究。我只希望这些观点可以给正在做 PL, 或者考虑进入这个领域的人们，作为一份参考。

PL 看似计算机科学最精髓的部分，事实确实也是这样的。没有任何一个其它领域，可以让你对程序的本质形成如此深入的领悟，然而这并不等于你就应该进入 PL 的博士班。这是为什么呢？

PL 这个领域几十年来，已经发展到了非常成熟的阶段。这里面的问题，要么在 20 年前已经被人解决掉了，要么就是类似 “停机问题” 一样，不可能解决的问题。然而，博士毕业却要求你发表“创新”的论文，那怎么办呢？于是你就只有扯淡，把别人已经解决的问题换个名字，或者制造一些看似新鲜却不管用的概念，在大会上煞有介事的宣讲。俗话说就是“炒冷饭”。

最开头进入这个领域的时候，你可能不觉得是这样，因为似乎有那么多的东西可以学习，那么多的大牛可以瞻仰，那么多的新鲜名词，什么 "lambda calculus" 啊，“语义” 啊，各种各样的 “类型系统” 啊，这样那样的 “逻辑”..... 可是时间久了，看透了，你就发现一些这个圈子里的规律。

几乎每篇 PL 领域的论文，里面必有一页弯弯曲曲，让人看花眼的逻辑公式。程序语言的论文，不是用程序来描述，而是用一些眼花缭乱的逻辑符号，像这样：

$$\text{(VAR)}\ \frac{\Gamma(x) = {}_\wedge T}{\Gamma \vdash^c x : {}_\wedge T} \qquad \text{(SUB)}\ \frac{\Gamma \vdash^c E : T \qquad T \leq T'}{\Gamma \vdash^c E : T'}$$

$$\text{(ABS}_{tp}\text{)}\ \frac{\Gamma, {}_\wedge\overline{a}, x : {}_\wedge T \vdash^c E : S}{\Gamma \vdash^c \mathbf{fun}[\overline{a}](x : T)E : {}_\wedge(T \xrightarrow{\overline{a}} \diamond S)} \qquad \text{(ABS)}\ \frac{\Gamma, {}_\wedge\overline{a}, x : {}_\wedge T \vdash^c E : S \qquad \overline{a} \notin \mathrm{tv}(E)}{\Gamma \vdash^c \mathbf{fun}(x)E : {}^\vee(T \xrightarrow{\overline{a}} \diamond S)}$$

$$\text{(APP}_{tp}\text{)}\ \frac{\Gamma \vdash^c F : {}^\vee({}_\wedge S \xrightarrow{\overline{a}} {}_\wedge T) \qquad \Gamma \vdash^c E : [\overline{R}/\overline{a}]^\vee S}{\Gamma \vdash^c F[\overline{R}](E) : [\overline{R}/\overline{a}]_\wedge T}$$

$$\text{(APP)}\ \frac{\begin{array}{c}\Gamma \vdash^c F : {}^\vee({}_\wedge S \xrightarrow{\overline{a}} {}_\wedge T) \qquad \Gamma \vdash^c E : S' \qquad S' \lhd_{\overline{a}} {}^\vee S \\ S' \leq [\overline{R}/\overline{a}]^\vee S \qquad [\overline{R}/\overline{a}]_\wedge T \leq T' \\ \forall \overline{R}', T''.(S' \leq [\overline{R}'/\overline{a}]^\vee S \ \wedge\ [\overline{R}'/\overline{a}]_\wedge T \leq T'' \sim T' \ \Rightarrow\ [\overline{R}/\overline{a}]_\wedge T \leq [\overline{R}'/\overline{a}]^\vee T)\end{array}}{\Gamma \vdash^c F(E) : T'}$$

$$\text{(SEL)}\ \frac{\Gamma \vdash^c E : {}^\vee\{x : \diamond T\}}{\Gamma \vdash^c E.x : T} \qquad \text{(REC)}\ \frac{\Gamma \vdash^c E_1 : \diamond T_1 \qquad \ldots \qquad \Gamma \vdash^c E_n : \diamond T_n}{\Gamma \vdash^c \{x_1 = E_1, \ldots, x_n = E_n\} : {}_\wedge\{x_1 : \diamond T_1, \ldots, x_n : \diamond T_n\}}$$

图 1.

绝大部分 PL 领域的专家们，似乎都酷爱逻辑符号，视逻辑学家高人一等。这种崇尚古人的倾向，使得 PL 专家们看不见这些符号背后，类似电路一样的直觉。他们看不见逻辑学的历史局限，所以他们也许能够发展和扩充一个理论，却无法创造一个新的。

说到古人，却并不是所有古人都这么晦涩。如果你考古一下就会发现，其实现代逻辑学的鼻祖 Gottlob Frege 最初的论文里，是没有这些稀奇古怪的符号的。他整篇论文都在画图，一些像电路一样的东西。比如下图，就是 Frege 的创始论文《Begriffsschrift》里最复杂的 “公式” 之一：

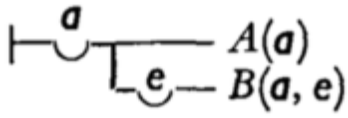


图 2.

你可以把这里的每根线理解成一根电线。图 1 里那些诡异的逻辑符号，都是一些好事的后人（比如 Gentzen）加进去的，最后搞得乌七八糟，失去了 Frege 理论的简单性。所以 PL 专家们虽然崇尚古人，却没发现大部分古人其实并没获得鼻祖 Frege 的真传。

如果你看透了那些公式，自己动手实现过各种解释器，就会发现 PL 论文里的那些公式，其实相当于解释器的代码，只不过是用一种叫做 "xx 逻辑" 的晦涩的语言写出来的。逻辑，其实是一种相当落伍的程序语言。如果你精通解释器的代码，也许就会发现，这些公式其实用蹩脚的方式，实现了哈希表等数据结构。

逻辑语言只运行于逻辑学家的脑子里面，用它写出的代码一样可能有 bug, 而且由于这语言如此障眼难读，而且没有 debugger, 所以 bug 非常难发现。逻辑学家们成天为自己的设计失误和 bug 伤透了脑筋，PL 专家们却认为他们具有数学的美感，是比自己聪明的高人。

所以当你看透了所有这些，就会发现 PL 的学术界，其实反反复复在解决一些早已经解决了的问题，只不过给它们起了不同的名字，使用不同的方式来描述。有时候好几个子领域，其实解决的是同一个问题，然而每个子领域的人，却都说自己的问题在本质上是不一样的，号称自己是那个子领域的鼻祖。甚至有人在 20 多年的时间里，制造出一代又一代的 PhD 和教授职位。他们的理论一代代的更新，最后却无法解决实际的问题。所谓的 “控制流分析” (control-flow analysis, CFA), 就是这样的一个子领域。

进入一个领域做研究，你总该知道那些人是真正厉害的。可惜的是，PL 这个领域里，你往往不知道谁是真正掌握了精髓的学者，甚至好几年之后你仍然蒙在鼓里。我的历史教训是，写教科书的人，往往不是最聪明，最理解本质的。真正深刻的 PL 研究者，你可能根本没听说过他们的名字。

一般程序员提到 PL，就会跟 “编译器” 这个领域混淆在一起，就会想起大学时候上编译器课，看《龙书》时焦头烂额的情景。然后由于斯德哥尔摩综合症，他们就会崇拜龙书的作者们。直到遇到了真正厉害的 PL 专家，你才发现编译器这个领域，跟 PL 根本是两回事，它其实比 PL 要低一个档次，里面充满了死记硬背的知识甚至误导。龙书的作者，其实也不是最厉害的编译器作者，他们更不是合格的 PL 专家。

上过 “正统” 的 PL 课程的学生，往往用一本经典大部头教材叫《TAPL》，然后就会误认为此书的作者是最厉害的 PL 专家，然而他们再一次被名气给蒙蔽了。TAPL 这书其实不但照本宣科， 没有揭示实质， 而且冗长没有选择， 有用的没用的过时的理论，一股脑的灌输给你。等你研究到了所谓 “交集类型” (intersection types), 看到 TAPL 作者当年的博士论文才发现， 其实他把简单的问题搞复杂了， 而且那些理论几乎完全不能实用。真正厉害的 intersection types 专家，其实默默无闻的待在 Boston University，而且研究到最后，intersection types 这个领域其实被他们证明为完全不能实用。

由于 TAPL 这本书，以及 ML，Haskell 等语言在 PL 界的 “白象” 地位，于是很多人又对 Hindley-Milner 类型系统（HM）充满了崇敬之情，以为 HM 系统的发明者 Robin Milner 是最厉害的 PL 学者。他的确不错，然而等你随手就能实现出 HM 系统，看清了它的实质，就会发现所有这样能够 “倒推” 出类型的系统，其实都具有很大的局限性。

HM 系统的 "unification" 机制，依赖于数学上的 “等价关系”， 所以它不可能兼容子类型 (subtyping) 关系。原因很简单： 因为子类型没有交换性， 不是一个等价关系。而子类型关系却是对现实世界进行直观的建模所必不可少的，于是你就发现 Haskell 这类基于 HM 系统的语言，为了弥补这些缺陷而出现各种 “扩展”， 却永远无法达到简单和直观。一开头就错了，所以无论 Haskell 如何发展，这个缺陷也无法弥补。如果没有了 HM 系统，Haskell 就不再是 Haskell.

Robin Milner 的另外一个贡献 π-calculus, 虽然看起来吓人，其实看透了之后你发现它里面并没有很多东西。π-calculus 对并发进行 “建模”，却不能解决并发所带来的各种问题，比如竞争 (race condition). 实际上普通的语言也能对并发进行简单的建模，所以 π-calculus 其实只停留于纸面上，不可能应用到现实中去。跟 π-calculus 类似的一个概念 CSP 也有类似的问题，属于 “白象理论”。 很多语言（比如 Go）扯着 CSP 的旗号，引起很多人无厘头的膜拜，可见白象的威力有多大。

我在学校研究 PL 的时候就是这样，每天都发现天外有天，每天都发现曾经的偶像其实很多时候是错觉。最后我发现，PL 领域其实最后就剩下那么一点点实质的内容，其它的都是人们造出来的浮云。所以每当有人问我推荐 PL 书籍，我都比较无语，因为我的 PL 知识只有非常少数是看书得来的。自己动手琢磨出来的知识，才是最管用的。

PL 的学生还有一个问题， 那就是毕业后工作不好找。 只有极少数公司（像微软，Intel，Oracle）里的少数团队，可以发挥 PL 专家的特殊才能。绝大部分其它公司根本不知道 PL 是什么，PL 专家是干什么的。你跟他们说你的专业是 “程序语言”， 他们还以为你只是学会了 “编程” 而已，还问你想做 “前端” 还是 “后端”。 诚然，PL 学生一般都有很好的编程能力，然而公司往往只关心自己的实际需求。PL 学生毕业之后，很容易被普通公司作为没有任何专长的人对待。

另外，PL 的圈子相当的小，而且门派宗教观念严重，所以就算你从名师手下毕业，想进入另一个老师的门徒掌权的公司， 很可能因为两个门派的敌视而无法被接纳， 就算进去了也经常会因为对于 PL 的理念不同而发生冲突。所以，学习 PL 最精髓的理论是有好处的，然而进入 PhD 投身 PL 的研究，我觉得应该三思。

PL 人在学校里跟着教授炒冷饭，毕业进入了公司之后，他们的行为方式还是非常类似。他们喜欢在公司里做的一件事情，叫做 “过度工程”。 本来很直接，很容易解决的一个问题，非要给你扯到各种炫酷的 PL 名词，然后用无比复杂的方案来解决。

有一些 PL 人喜欢推广他们认为高大上的语言，比如 Haskell, OCaml, Scala 等。这些语言在 PL 学术界很受尊重，所以他们以为这些语言能够奇迹般的解决实际的问题，然而事实却不是这样的。事实是，这些学术界出来的语言，其实缺乏处理现实问题的机制。为了能够在学术上证明程序的所谓 “正确性”， 而且由于类型系统本身的局限性，这些语言往往被设计得过于简单，具有过度的约束性，以至于表达能力欠缺。

最后，你发现用这些语言来写代码，总是这也不能做，那也不能做，因为你要是那么做了，编译器就无法发现“类型错误”。到最后你发现，这些语言的约束，其实是无需有的。如果放宽这些约束，其实可以更优雅，更简单的对问题进行建模。对正确性的过分关注，其实导致了 PL 人选择蹩脚的语言，写出绕着弯子，难以理解的代码。

还有一类 PL 人，喜欢设计不必要存在的语言。因为他们认为设计语言是 PL 人的特异功能，所以随时随地都想把问题往 “语言设计” 的方向上靠。这样的趋势是非常危险的，因为有原则的 PL 人，其实都明白一条重要的道理：不到万不得已的时候，千万不要制造语言。

很多 PL 人在公司里盲目的制造新的语言，导致的问题是，到最后谁也无法理解这种新语言写出来的代码。这一方面是新语言必然导致的结果，另一方面是由于，并不是每一个 PL 人都有全面的知识和很好的 “品味”。 每个 PL 学生毕业，往往只深入研究了 PL 的某个子领域，而对其它方面只是浮光掠影，所以他们有可能在那上面犯错。

有些 PL 人喜欢照猫画虎，所以可能盲目的模仿 Go 语言，Haskell 或者 Python 的特性，设计出非常蹊跷难用的语法。这些新的语言，其实让其他人苦不堪言。最后你发现，他们声称新语言能解决的问题，其实用像 Java 一样的老语言，照样可以很容易的解决。

喜欢钻牛角尖，把问题搞复杂，就是很多公司里的 PL 人的共同点。制造语言是 PL 人应该尽量避免的事情，这恰恰跟 PL 人的专长是矛盾的。所以有原则的 PL 人，生活怎么可能不苦。

很多研究 PL 的人喜欢看低其它程序员，认为自己能设计实现程序语言，就是天之骄子。我之所以从 Dan Friedman 那里学到了好东西，却没有成为他的 PhD 学生，一方面就是因为看不惯围绕在他身边那些自认为是 “天才” 的人。

总是有那么一群本科生，自认为掌握了 Friedman 所讲授的精髓，所以高人一等。于是我就经常无奈的看着他们，吵吵闹闹的宣讲他们解决的“新问题”，貌似什么了不起的发明一样，受到 Friedman 的肯定就受宠若惊的样子。而其实呢，那些都是我几年前就已经试过并且抛弃的方案。

其它的 PL 人，包括 PhD 学生，也有一样的毛病。不管在三流大学，还是在 Harvard, Princeton, MIT 这样的 “牛校” 出来的，只要是 PL 人，几乎必然有这种天才作风。另外你可能不知道的是，牛校往往并不产出优秀的 PL 人才。像 Stanford, Berkeley, MIT 这样的传统 CS 牛校，其实在 PL 方面是相当差的。

这种天才病的危害在于，它蒙蔽了这些人的眼睛。他们不再能设计出让“普通人”可以容易使用的产品。如果你不会用，他们就会嘲笑你笨，而其实呢，是因为他们的设计不好。他们喜欢用含混晦涩的方式（所谓 “函数式”）的写法来构造代码，让其它人阅读和修改都极其困难。

这些所谓天才，看不到简单直观的解决方案，为了显示自己的聪明而采用繁复的抽象，其实是一种愚蠢。真正的天才，必须能够让事情变得简单。

2 Hindley-Milner 类型系统的根本性错误

之前的一个时间， 我曾经公开过这样一段幻灯片， 它是2012年10月的时候， 我在 Indiana 大学做的最后一次演讲。 由于当时的委婉， 我并没有直接说出这些结论的重要性。 其实它揭示了 ML 和 Haskell 这类基于 Hindley-Milner 类型系统的语言的一个根本性的错误。

这个错误来源于对一阶逻辑的 “全称量词”（universal quantifier，通常写作 ∀）与程序函数之间关系的误解。在 HM 系统里面，多态（polymorphic）的函数能够被推导为含有全称量词的类型，比如 \x->x 的类型被推导为 ∀a.a->a, 但 HM 系统决定这个全称量词的位置的方式，却是没有原则的。这就导致了类型变量（type variable）的作用域（scope）的偏差。

我的研究显示， 这个错误来源于 HM 系统最初的一项重要的设计， 叫做 let-polymorphism。如果右边的函数是一个多态的函数，比如：

```
let f = \x->x in
...
```

let-polymorphism 总是会把全称量词的位置确定在 let 的 “=” 右边。然而这是一个非常错误的做法，它的错误程度近似于早期的 Lisp 所采用的 dynamic scoping. 这样做的结果是，全称量词的位置会随着程序的 “格式”， 而不是程序的 “结构”， 而变化。

为了弥补这个错误，30多年来，许多的人发表了许多的论文，提出了很多的“改进措施”，比如 value restriction，MLF，等等。但是我的研究却显示，所有这些“改进措施”都是丑陋的 hack。因为他们没有看到问题的根源，所以他们的方案只对一些特殊情况起作用， 而不能通用。 为此，我可以轻而易举的写出正确的程序， 而让它不能通过这些类型系统的检查，比如像我这篇英文博文所示。如果你看到了问题的根源，就会发现 HM 系统的这个错误是无法弥补的， 因为它触及了 HM 系统的根基。 为了根治这个问题，let-polymorphism 必须被去除掉。

我为此提出了自己的解决方案：在 lambda 的位置进行 "generalization", 也就是说把 ∀ 放在 lambda 的位置，而不是 let. 这样一来 let-polymorphism 就不存在了。但是这样一来，HM 系统就不再是 HM 系统，因为它的 “模块化类型推导” 的性质，就会名存实亡。由于类型里面含有程序的 “控制结构”， 这个类型系统表面上看起来是在进行 “模块化类型检查”， 而本质上是在做一个 “跨过程静态检查” (interprocedual static analysis). 也就是说，模块化的类型推导，在 HM 这样的没有 “类型标记” 的体系下，其实是不可能实现的。

为了达到完全通用的模块化类型检查， 却又允许多态函数的存在， 我们终究会需要在函数的参数位置手工写上类型，这样我们就完全的丧失了 HM 系统设计的初衷。

3 编辑器与IDE

很多人都喜欢争论哪个编辑器是最好的。 其中最大的争论莫过于 Emacs 与 vi 之争。vi 的支持者喜欢说：“看 vi 打起字来多快，手指完全不离键盘，连方向键都可以不用。” Emacs 的支持者往往对此不屑一顾，说：“打字再快又有什么用。我在 Emacs 里面按一个键，等于你在 vi 里面按几十个键。”

其实还有另外一帮人， 这些人喜欢说：“对于 Emacs 与 vi 之争， 我的答案是 {jEdit， Geany, TextMate, Sublime...}”这些人厌倦了 Emacs 的无休止的配置和 bug, 也厌倦了 vi 的盲目求快和麻烦的模式切换， 所以他们选择了另外的更加简单的解决方案。

那么我对此的答案是什么呢？在目前的情况下，我对程序编辑的临时答案是：IDE.

写程序的时候，我通常根据语言来选择最能 “理解” 那种语言的 IDE（比如 Visual Studio, Eclipse, IntelliJ IDEA 等），而不是一种通用的 “文本编辑器”（比如 Emacs, vi, jEdit, ...）。这是因为 “文本编辑器” 这种东西一般都不真正的理解程序语言。很多 Emacs 和 vi 的用户以为用 etags 和 ctags 这样的工具就能让他们 “跳转到定义”， 然而这些 tags 工具其实只是对程序的 “文本” 做一些愚蠢的正则表达式匹配。它们根本没有对程序进行 parse, 所以其实只是在进行一些 “瞎猜”。 简单的函数定义它们也许能猜对位置，但是对于有重名的定义，或者局部变量的时候，它们就力不从心了。

很多人对 IDE 有偏见，因为他们认为这些工具让编程变得 “傻瓜化” 了，他们觉得写程序就是应该 “困难”， 所以他们眼看着免费的 IDE 也不试一下。有些人写 Java 都用 Emacs 或者 vi, 而不是 Eclipse 或者 IntelliJ. 可是这些人错了。他们没有意识到 IDE 里面其实蕴含了比普通文本编辑器高级很多的技术。这些 IDE 会对程序文本进行真正的 parse, 之后才开始分析里面的结构。它们的 “跳转到定义” 一般都是很精确的跳转，而不是像文本编辑器那样瞎猜。

这种针对程序语言的操作可以大大提高人们的思维效率， 它让程序员的头脑从琐碎的细节里面解脱出来， 所以他们能够更加专注于程序本身的语义和算法， 这样他们能写出更加优美和可靠的程序。这就是我用 Eclipse 写 Java 程序的时候相对于 Emacs 的感觉。我感觉到自己的 “心灵之眼” 能够 “看见” 程序背后所表现的 “模型”， 而不只是看到程序的文本和细节。 所以，我经常发现自己的头脑里面能够同时看到整个程序， 而不只是它的一部分。 我的代码比很多人的都要短很多也很有很大部分是这个原因， 因为我使用的工具可以让我在相同的时间之内， 对代码进行比别人多很多次的结构转换， 所以我往往能够把程序变成其他人想象不到的样子。

对于 Lisp 和 Scheme, Emacs 可以算是一个 IDE. Emacs 对于 elisp 当然是最友好的了，它的 Slime 模式用来编辑 Common Lisp 也相当不错。然而对于任何其它语言，Emacs 基本上都是门外汉。我大部分时间在 Emacs 里面是在写一些超级短小的 Scheme 代码，我有自己的一个简单的配置方案。虽然谈不上是 IDE, Emacs 编辑 Scheme 确实比其它编辑器方便。R. Kent Dybvig 写 Chez Scheme 居然用的是 vi, 但是我并不觉得他的编程效率比我高。 我的代码很多时候比他的还要干净利落， 一部分原因就是因为我使用的 ParEdit mode 能让我非常高效的转换代码的 “形状” 。

当要写 Java 的时候，我一般都用 Eclipse. 最近写 C++ 比较多，C++ 的最好的 IDE 当然是 Visual Studio. 可惜的是 VS 没有 Linux 的版本，所以就拿 Eclipse 凑合用着，感觉还比较顺手。个别情况 Eclipse “跳转定义” 到一些完全不相关的地方，对于 C++ 的 refactor 实现也很差，除了最简单的一些情况（比如局部变量重命名），其它时候几乎完全不可用。当然 Eclipse 遇到的这些困难，其实都来自于 C++ 语言本身的糟糕设计。

想要设计一个 IDE, 可以支持所有的程序语言，这貌似一个不大可能的事情，但是其实没有那么难。有一种叫做 “结构化编辑器” 的东西，我觉得它可能就是未来编程的终极解决方案。

跟普通的 IDE 不同，这种编辑器可以让你直接编辑程序的 AST 结构，而不是停留于文本。每一个界面上的 “操作”， 对应的是一个对 AST 结构的转换，而不是对文本字符的 “编辑”。 这种 AST 的变化，随之引起屏幕上显示的变化，就像是变化后的 AST 被 "pretty print" 出来一样。 这些编辑器能够直接把程序语言保存为结构化的数据（比如 S-表达式，XML 或者 JSON），到时候直接通过对 S-表达式，XML 或者 JSON 的简单的 “解码”， 而不需要针对不同的程序语言进行不同的 parse. 这样的编辑器，可以很容易的扩展到任何语言， 并且提供很多人都想象不到的强大功能。 这对于编程工具来说将是一个革命性的变化。

我之前推荐过的 TeXmacs 其实在本质上就是一个 “超豪华” 的结构化编辑器。你可能不知道，TeXmacs 不但能排版出 TeX 的效果，而且能够运行 Scheme 代码。IntelliJ IDEA 的制造者 JetBrains 做了一个结构化编辑系统，叫做 MPS. 它是开源软件，并且可以免费下载。 另外， Microsoft Word 的创造者 Charles Simonyi 开了一家叫做 Intentional Software 的公司，也做类似的软件。

4 程序语言，操作系统，数据库三位一体

如果 main 函数可以接受多种类型的参数，并且可以有 keyword argument, 它能返回一个或多个不同类型的对象作为返回值， 而且如果这些对象可以被自动存储到一种特殊的 “数据库” 里，那么 shell，管道，命令行选项，甚至连文件系统都没有必要存在。我们甚至可以说，“操作系统” 这个概念变得 “透明”。 因为这样一来，操作系统的本质不过是某种程序语言的 “运行时系统” (runtime system). 这有点像 JVM 之于 Java。其实从本质上讲，Unix 就是 C 语言的运行时系统。

如果我们再进一步，把与数据库的连接做成透明的，即用同一种程序语言来 “隐性” (implicit) 的访问数据库，而不是像 SQL 之类的专用数据库语言，那么 “数据库” 这个概念也变得透明了。我们得到的会是一个非常简单，统一，方便，而且强大的系统。这个系统里面只有一种程序语言， 程序员直接编写高级语言程序， 用同样的语言从命令行执行它们， 而且不用担心数据放在什么地方。 这样可以大大的减小程序员工作的复杂度，让他们专注于问题本身，而不是系统的内部结构。

实际上，类似这样的系统在历史上早已存在过 (Lisp Machine, System/38, Oberon), 而且收到了不错的效果。但是由于某些原因（历史的，经济的，政治的，技术的），它们都消亡了。但是不得不说它们的这种方式比 Unix 现有的方式优秀，所以何不学过来？我相信，随着程序语言和编译器技术发展， 它们的这种简单而统一的设计理念， 有一天会改变这个世界。

5 原因与证明

我在 Cornell 的时候经常遇到这样的问题，那就是教授们一上课就在黑板上写长篇的 “定理证明”， 全体同学认认真真在下面抄笔记， 就连只有十来个人的小课也是那样。有些写字速度慢的人就不得不带上小型录音机， 把教授的课全都录下来， 要不就是之后去借别人的笔记来抄。

有一次某知名教授照着讲义，背对着学生，在黑板上写了大半节课，写下好几板的证明，证明的是 simply typed lambda calculus (STLC) 的 strong normalization 特性（SN）。刚写完就到下课时间了，他回过头来喘了一口气，说："Any questions?" 没有人啃声，于是他说："很好！下课！"

几天后我问他，你证明了 STLC 有这个特性，然而你却没有告诉我它 "为什么" 有这个特性。他神气的看了我一眼："你不懂吗？" 我说："你的证明我看懂了大部分，可是一个东西具有如此的性质，并不是因为你证明了它。这性质是它天生就有的，不管你是否能证明它。我想知道的是什么让 STLC 具有这个性质，而不只是证明它。" 他说："你问这样的问题有什么意义吗？你需要非常聪明，并且需要经过大量的努力才能想出这样的证明。"

两年之后，我在 Indiana 上了另外一堂程序语言理论课。教授是我之前的导师 Amr Sabry. 他上课从来不带讲义，貌似也没有准备，漫不经心的，却每次都能讲清楚问题的关键。于是有一天他也开始讲 STLC 的 SN 特性。他说，我不想写下这个证明让你们抄，我只告诉你们大概怎么去想。SN 的意思就是程序肯定会 "终止"。所有会终止的程序，都会有一个"特征值"会随着程序的运行而减小。你需要做的就是找到 STLC 的 "特征值" 是什么。接着他就开始在黑板上画一个图.....

过了一段时间，我不仅学会了这个"证明"，而且知道了 STLC 具有如此特性的 "原因"。

从以上的故事，以及你的亲身经历中，你也许注意到了大部分的教育过分的重视了 "证明"，却忽略了比证明更重要的东西 —— "原因"。

原因往往比证明来得更加简单，更加深刻，但却更难发现。对于一个事实往往有多种多样的证明，然而导致这个事实的原因却往往只有一个。如果你只知道证明却不知道原因，那你往往就被囚禁于别人制造的理论里面，无法自拔。你能证明一个事物具有某种特性，然而你却没有能力改变它。你无法对它加入新的，好的特性，也无法去掉一个不好的特性。你也无法发明新的理论。有能力发明新的事物和理论的人，他们往往不仅知道 "证明"，而且知道 "原因"。

打个比方。证明与原因的区别，就像是犯罪的证据与它的原因的区别。证据并不是导致犯罪的原因。有了证据可以帮助你把罪犯绳之以法，可是如果你找不到他犯罪的原因，你就没法防止同样的犯罪现象再次发生。

古人说的 "知其然" 与 "知其所以然" 的区别，也就是同样的道理吧。

6 经验和洞察力

很多人很在乎 "经验"，比如号称自己在某领域有 30 年的经验，会用这样那样的技术。我觉得经验是有价值的，我也有经验，各个领域的都有点。然而我并不把经验放在很重要的位置，因为我拥有大部分人都缺乏而且忽视的一种东西：洞察力（insight）。

每进入一个新的公司，我进入的几乎都是不同的领域。所以最开头的时候，我有可能对那个领域所知甚少。甚至有人觉得我没有经验，所以可以 "教育" 我。然而每一次他们都没有想到的是，我很快就掌握了他们的经验，并且经过提炼，抛弃其中的垃圾，很快的超越了他们，完成他们根本无法达到的目标。这就是洞察力的威力。

举个亲身例子，很多人都有用线程的经验，可是有多少人知道线程的本质是什么？有多少人在头脑里有一幅画面，显示出多线程程序的各种动态特征？其实很少有人知道。这就是为什么很多人过度的使用线程并发，结果产生各种同步问题，竞争状态（race condition），死锁等现象。某公司的一片多线程代码，号称是“有非常多并发程序经验”的程序员写的。结果没多久我就发现里面其实含有非常微妙的竞争情况，会在非常小的概率随机发作。发现之后没过几天，已经卖出去用了两年多的产品，由于这个竞争情况，终于引发了严重的后果。有那么多并发编程经验的程序员，两年多都没有察觉这个竞争情况，而很少写多线程程序的我，不但发现了这个竞争，而且很快的想出了修复它的办法，这是为什么呢？靠的就是洞察力。我知道线程的本质，而这是经验不会告诉你的。

什么是洞察力？洞察力就是透过现象看到本质的能力。有洞察力的人很容易得到经验，然而有经验的人却不一定有洞察力。再愚钝的人，总是可以通过大量的时间获取经验，然而就算你花再多的时间和精力，也难以得到洞察力。所以洞察力是比经验宝贵很多的东西。很难说清楚如何才能有洞察力，也很少有人会告诉你如何去得到它。当然，我也不会告诉你。

看别人简历，经常会列出各种各样的技术经验，我看一眼就会的东西，也会在上面占个位置。由于这个原因，我把自己 Linkedin 上面曾经列出的 “工作经验” 全都删掉了。这些东西列在那里，对于我本身的价值，实在是一种贬低。我是一个身上不贴任何标签的，不能被任何头衔所局限的，真正有价值的人。

经验虽然不是最重要的，然而还是有必要的。很多技术你不能完全不碰它，然而一碰就明白了。但如果没有实际的问题，你又会没有动力去接触那些技术。所以我一直在做的一件事情，就是接触各种技术，然后利用洞察力来获得越来越多的经验。回国之后的初期，我打算着手做自己的产品。同时，我想跟国内的各种公司或者个人做这样的交易。我利用洞察力帮助解决他们最棘手的，已有经验无法解决的难题，从而让我获得经验。当然，我不是作为公司的职工，而只是作为独立的顾问。对公司我会象征性的收取一定的费用，换句话，就是作为 “职业杀手”。对于个人，他的问题必须对我也有启发意义。对此感兴趣的公司或者个人，可以跟我联系。

7 创造者的思维方式

我不知道人们是怎么回事，缺乏想象力还是怎么的，所以我跟其他人对话常常遇到类似的问题。

我：A 其实不怎么好。
其他人：你说 A 不好，难道你要我用B？

（对于政治爱好者，如果 A 是资本主义，B 就是社会主义；如果 A 是美国，B 就是中国，等等。对于 IT 人员，如果 A 是 Unix, B 就是 Windows; 如果 A 是 Vim, B 就是 Emacs; 如果 A 是关系式数据库，B 就是 NoSQL 数据库，等等.....）

然后呢，这人就会深信我是 B 的拥蠓，进而产生敌意。这种对话越说越糊涂，越说越尴尬，越说我越觉得降低我的身份。

仔细分析之后，我发现了这问题的起因，其实是因为我跟其他人的思维方式是完全不同的。我总是从一个 “创造者” 的角度说话，而对方却站在 “使用者” 的角度。站的高度不同，当然就没法沟通，鸡同鸭讲。

创造者说 “A其实不怎么好”， 他的意思往往不是说你应该去 “用” 别的什么东西。他的意思其实是，A不怎么好，我可以把它的缺点去掉， “创造” 一个更好的东西。这里的区别就在于 “用” 和 “创造” 的不同。使用者说 “A 不好”， 是无可奈何的抱怨；创造者说 “A 不好”， 却是对改进机遇的欣喜。可惜的是，使用者永远无法理解创造者的心，创造者的喜悦在使用者的头脑里，直接被 “翻译” 成了抱怨。

创造者拥有使用者没有的能力，他能够随心所欲的制造出新的事物，而不带有现存事物设计的思维枷锁。创造者因此具有比使用者更高的安全感，更深的远见，更豁达的胸襟。他不容易陷入非此即彼的 “宗教冲突”， 他不需要选择任何一个 “阵营”， 因为他对这种冲突的解决方案很简单：创造一个全新的宗教，消灭掉冲突的双方 :)

8 机器与人类视觉能力的差距（1）

很多人以为人工智能就快实现了，往往是因为他们混淆了 “识别” 和 “理解”。 现在所谓的 “人工智能” 都是在做识别：语音识别，图像识别，而真正的智能是需要理解能力的。我们离理解有多远呢？恐怕真正的工作根本就没开始。

很长时间以来，我都在思索理解与识别的差别。理解与识别是很不一样的，却总是被人混为一谈。我深刻的明白理解的重要性，可是我发现很少有其他人知道 “理解” 是什么。AI 领域因为混淆了识别和理解，一直以来处于混沌之中。

最近因为图像识别等领域有了比较大的进展，人们对 AI 产生了很多科幻似的，盲目的信心，出现了自 1980 年代以来最大的一次 “AI 热”。 很多人以为 AI 真的要实现了，被各大公司鼓吹的 “黑科技” 冲昏了头脑，却看不到现有的 AI 方法与人类智能之间的巨大鸿沟。 所以下面我想介绍一下我所领悟到的机器和人类在视觉能力方面的差距，希望一些人看到之后，能够再次拥有冷静的头脑。

在之前一篇文章《人工智能的局限性》中，我已经阐述了对自然语言处理领域误区的看法。当时因为对计算机视觉方面了解不多，所以没有包含视觉方面的内容。熟悉了机器视觉的各种做法之后，我想在这篇文章里详述一下视觉方面的内容。这两篇文章加在一起，可以说概括了我对 AI 语言和视觉两个方面的领悟。

对于视觉，AI 领域混淆了 “图像识别” 和 “视觉理解”。 现在热门的所谓 "AI" 都是 “图像识别”， 而动物的视觉系统具有强大的 “视觉理解”。 视觉理解和图像识别有着本质的不同。

深度学习视觉模型（CNN 一类的）只是从大量数据拟合出从 “像素=>名字” 的函数。它也许能从一堆像素猜出图中物体的 “名字”， 但它却不知道那个物体 “是什么”， 无法对物体进行操作。注意我是特意使用了 “猜” 这个字，因为它真的是在猜，而不像人一样准确的知道。

“图像识别” 跟 “语音识别” 处于同样的级别，停留在语法（字面）层面，而没有接触到 “语义”。 语音识别是 “语音=>文字” 的转换，而图像识别则是 “图像=>文字” 的转换。两者都输出文字，而 “文字” 跟 “理解” 处于两个不同的层面。文字是表面的符号，你得理解了它才会有意义。

怎样才算是 “理解了物体” 呢？至少，你得知道它是什么形状的，有哪些组成部分，各部分的位置和边界在哪里，大概是什么材料做成的，有什么性质。这样你才能有效的

对它采取行动，达到需要的效果。否则这个物体只是一个方框上面加个标签，不能精确地进行判断和操作。

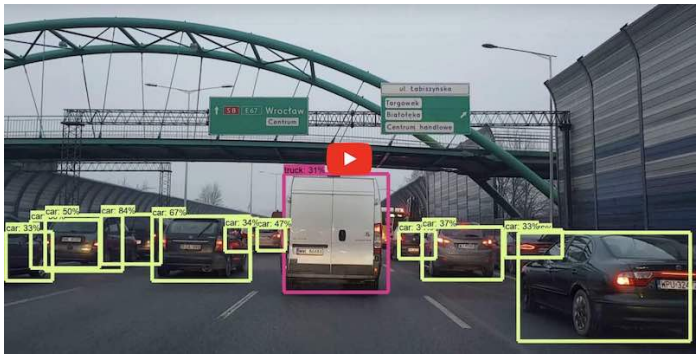


图 3.

想想面对各种日常事物的时候，你的脑子里出现的是它们的名字吗？比如你拿起刀准备切水果，旁边没有人跟你说话，你的脑子里出现了“刀”这个字吗？一般是没有的。你的脑子里出现的不是名字，而是“常识”。常识不是文字，而是一种抽象而具体的数据。

你知道这是一把刀，可是你的头脑提取的不是“刀”这个字，而是刀“是什么”。你的视觉系统告诉你它的结构是什么样的。你知道它是金属做的，你看到刀尖，刀刃，刀把，它也许是折叠的。经验告诉你，刀刃是锋利的可以切东西的部分，碰到可能会受伤，刀把是可以拿的地方。如果刀是折起来的，你得先把它翻开，那么你从哪一头动手才能把它翻开，它的轴在哪里？

你顺利拿起刀，开始切水果。可是你的头脑里仍然没有出现“刀”这个字，也没有“刀刃”，“刀把”之类的词。在切水果的同时，你大脑的“语言中心”可能在哼一首最近喜欢的歌词，它跟刀没有任何关系。语言只是与其他人沟通的时候需要的工具，自己做事的时候我们并不需要语言。完成切水果的动作，你需要的是由视觉产生的对物体结构的理解，而不是语言。

你不需要知道一个物品叫什么名字就能正确使用它。同样的，光是知道一个物品的名字，并不能帮助你使用它。看到一个物体，如果脑子里首先出现的是它的名字，那么你肯定是很愚钝的人，无法料理自己的生活。现在的“机器视觉”基本就是那样的。机器也许能得出图片上物体的名字，却不知道它是什么，无法操作它。

试想一下，一个不能理解物体结构的机器人，它只会使用图像识别技术，在你的头上识别出一个个的区域，标注为“额头”，“头发”，“耳朵”..... 你敢让它给你理发吗？

这就是我所谓的“视觉理解”与“图像识别”的差别。你会意识到，这种差别是巨大的。

如果我们降低标准，只要求识别出物体的名字，那么以像素为基础的图像识别，比如卷积神经网络（CNN），也是没法像人一样准确识别物体的。人识别物体并不是像神经网络那样的“拍照，识别”两节拍动作，而是一个动态的，连续的过程：观察，理解，观察，理解，观察，理解.....

感官接受信息，中间穿插着理解，理解反过来又控制着观察的方向和顺序。理解穿插在了识别物体的过程中，“观察/理解”成为不可分割的整体。人看到物体的一部分，理解了那是什么，然后继续观察它周围是什么，反复这个过程，最后才判断出物体是什么。机器在识别的过程中没有理解的成分存在，这就是为什么机器在图像识别能力上无

法与人类匹敌。

这个 “观察/理解” 的过程发生的如此之快，眨眼间就完成了，以至于很多人都没察觉到其中 “理解成分” 的存在。所以我们现在放慢这个过程，来一个慢镜头特写，看看到底发生了什么。假设你从来没见过下面这个东西，你知道它是什么吗？

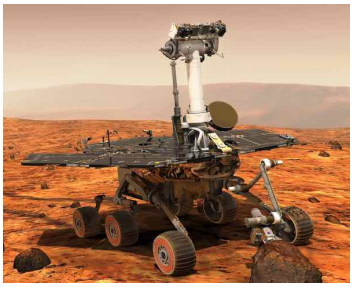


图 4.

一个从没见过这东西的人，也会知道这是个 “车”。为什么呢？因为它有轮子。为什么你知道那是轮子呢？仔细一想，因为它是圆的，中间有轴，所以好像能在地面上滚动。为什么你知道那是 “轴” 呢？我就不继续折腾你了，自己想一下吧。所有这些分析都是 “视觉理解” 所产生的，而这些理解依赖于你一生积累的经验，也就是我所谓的 “常识”。

其实为了识别这个东西，你并不需要分析这么多。你之所以做这些分析，是因为另一个人问你 “你怎么知道的？” 人识别物体靠的是所谓 “直觉”。一看到这个图片，你的脑子里自然产生了一个 3D 模型。一瞬间之后，你意识到这个模型符合 “车” 的机械运动原理，因为你以前看见过汽车，火车，拖拉机..... 你的脑子里浮现出这东西可能的运动镜头，你仿佛看到它随着轮子在动。你甚至看到其中一个轮子压到岩石，随着连杆抬了起来，而整个车仍然保持平衡，没有反倒，所以这车也许能对付崎岖的野外环境。

这里有一个容易忽视的要点，那就是轮子的轴必须和车体连在一起。如果轮子跟车体没有连接，或者位置不对，看起来无法带着车体一起运动，人都是知道的。这种轮轴与车身的连接关系，属于一种叫 “拓扑” (topology) 的概念。

拓扑学是一门难度挺高的数学分支，但人似乎天生就理解某些浅显的拓扑概念。实际上似乎高等动物都或多或少理解一些拓扑概念，它们一看就知道哪些东西是连在一起的，哪些是分开的。捕猎的动物都知道，猎物的尾巴是跟它们身体连在一起的，所以咬住它们的尾巴就能抓住它们。

拓扑学还有一个重要的概念，那就是 “洞”。聪明一点的动物基本上都理解 “洞” 的概念。很显然老鼠，兔子等穴居动物必须理解洞是什么。它们的天敌，猫科动物等，也理解洞是什么。如果我拿一个纸箱给我的猫玩，我在上面挖一个洞，等他钻进去，他是不会进去的。我必须在上面挖两个洞，他才会进去。为什么呢？因为他知道，要是箱子上面只有一个洞，要是他进去之后洞被堵上，他就出不来了！

机器如何才能理解洞这个概念呢？它如何理解 “连续” ？

总之，人看到物体，他看到的是一个 3D 模型，他理解其中的拓扑关系和几何性质，所以一个人遇到前所未见的物体，他也能知道它大概是什么，推断出如何使用它。理解使得人可以非常准确地识别物体。没有理解能力的机器是做不到这一点的。

人的眼睛与摄像头有着本质的差异。眼睛的视网膜中央非常小的一块区域叫做 "fovea", 里面有密度非常高的感光细胞，而其它部分感光细胞少很多，是模糊的。可是眼睛是会转动的，它被脑神经控制，敏捷地跟踪着感兴趣的部分：线条，平面，立体

结构..... 人的视觉系统能够精确地理解物体的形状，理解拓扑，而且这些都是 3D 的。人脑看到的不是像素，而是一个 3D 拓扑模型。

眼睛观察的顺序，不是一行一行从上往下把每个“像素”都记下来，做成 6000×4000 像素的图片，而是聚焦在重点上。它可以沿着直线，也可以沿着弧线观察，可以转着圈，也可以跳来跳去的。人脑通过自己的理解能力，控制着眼睛的运动，让它去观察所需要的重点。由于视网膜中央分辨率极高，所以人脑可以得到精度非常高的信息。然而由于不是每个地方都看的那么仔细，所以眼睛采集的信息量可能不大，人脑需要处理的信息也不会很多。

人的视觉系统能理解点，线，面的概念，理解物体的表面是连续的还是有洞，是凹陷的还是凸起的，分得清里和外，远和近，上下左右..... 他能理解物体的表面是什么质地，如果用手去拿会有什么样的反应。他能想象出物体的背面大概是什么样子，他能在头脑中旋转或者扭曲物体的模型。如果物体中间有缺损，他甚至能猜出那位置之前什么样子。

人的视觉系统比摄像头有趣的多。很多人都看过“光学幻觉” (optical illusion) 的图片，它们从一个角度揭示了人的视觉系统背后在做什么。比如下图本来是一个静态的图片，可是你会感觉有很多暗点在白线的交叉处，但如果你仔细看某一个交叉处，暗点却又不见了。这个幻觉很经典，被叫做 Herman grid, 在神经科学界被广泛研究。稍后我还会提到这个东西。

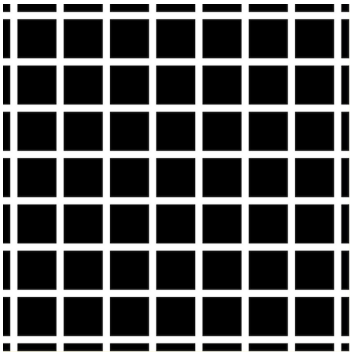


图 5.

本来是静态图片，你却感觉它在转。

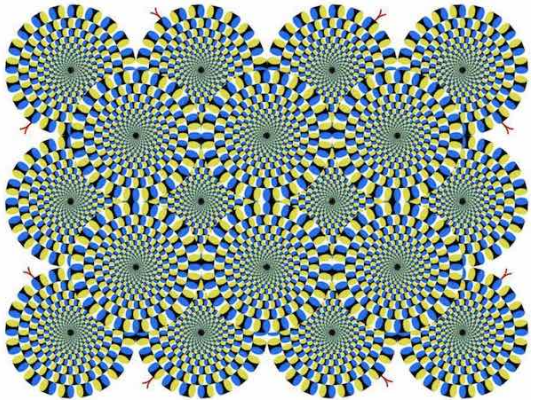


图 6.

本来上下两块东西是一样的颜色，可是看起来下面的颜色却要浅一些。如果你用手指挡住中间的高亮部分，就会发现上下两块的颜色其实是一样的。

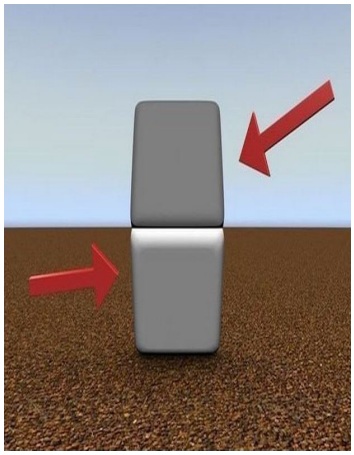


图 7.

另一个类似的幻觉，是著名的 "Abelson 棋盘幻觉". 图中 A 和 B 两个棋盘格子的颜色是一样的，你却觉得 A 是黑色，而 B 是白色。不信的话你可以用软件把这两块格子从图片上切下来，挨在一起对比一下。

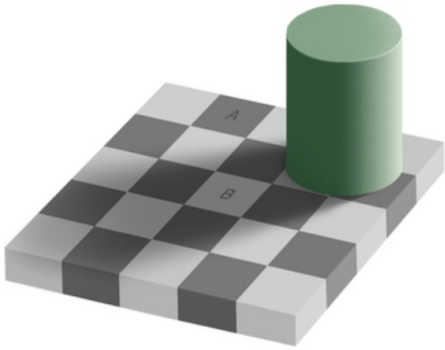


图 8.

在下图里，你会觉得看见了一个黑色的倒三角形，可是其实它并不存在。

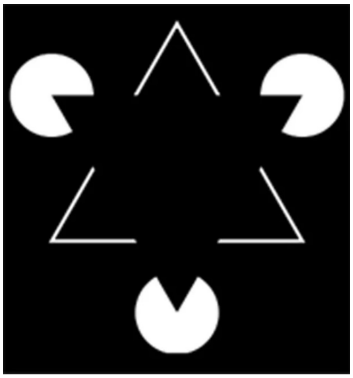


图 9.

很多的光学幻觉都说明人的视觉系统不是简单的摄像头一样的东西，它具有某些特殊功能。这些特殊功能和机制导致了这些幻觉。这使得人类视觉不同于机器，使得人能够提取出物体的结构信息，而不是只看到像素。

提取物体的拓扑结构特征，这就是为什么人可以理解抽象画，漫画，玩具。虽然世界上没有猫和老鼠长那个样子，一个从来没看过《猫和老鼠》动画片的小孩，却知道这是一只猫和一只老鼠，后面有个房子。你试试让一个没有拿《猫和老鼠》剧照训练过的深度学习模型来识别这幅图？

图 10.

更加抽象的玩具，人也能识别出它们是哪些人物。头和四肢都变成了方的，居然还是觉得很“像”。 你不觉得这很神奇吗？

图 11.

人脑理解 “拓扑” 的概念，这使得人能够不受具体像素干扰而正确处理各种物体。对拓扑结构的理解使得人对物体的识别非常准确，甚至可以在信息不完整，模糊，扭曲的情况下工作，在恶劣的天气环境下，有反光，有影子的情况下也能识别物体。

说到反光，你有想过机器要如何才能识别出场景里有一面镜子或者玻璃吗？如果场景中有反光的物体，比如镜子，平静的水面，镀铬的物品，神经网络（CNN）那种依靠像素滤镜训练出来的函数还会有用吗？要知道它们看到的像素，可能有一大片是通过镜面反射形成的，所以无法通过局部的纹理识别出这种情况来。

图 12.

这是个现实的问题。自动车或者机器人要如何知道前面的路面上有积水或者结冰了？它们要如何知道从水面反射过来的镜像不是真实的物体？比如，它们如何知道下图里路面上的倒影不是真正的树呢？要知道，倒影的像素纹理，跟真实的场景可能是非常相似的。

图 13.

人是通过对光的理解，各种常识来识别镜子，玻璃，地上的水和冰的存在。一个不理解光和水的性质的机器，它能察觉这些东西的存在吗？靠像素分析能知道这些？要知道，这些东西在某些地方出现，可以是致命的危险。

很有趣的事情，理解光线的反射和折射，似乎已经固化到了每个动物的视觉系统里面。我观察到这一点，是因为我的卧室和客厅之间的橱柜门上有两面大镜子。我的猫在卧室里，能够从镜子里看见我在客厅拿着逗猫绳。他冲过来的时候却不会撞到镜子上面，

而是出了卧室门立马转一个角度， 冲向我的方向。 我每次看到他敏捷的动作都会思考，他是如何知道镜子的存在呢？他是如何知道镜子里的猫就是他自己，而不是另一只猫？

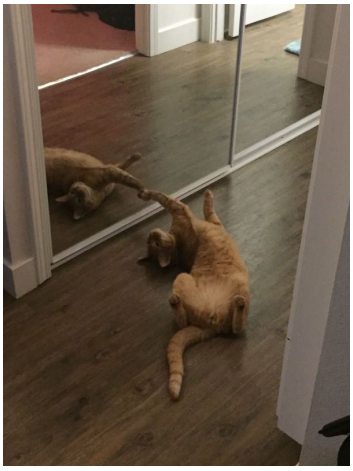


图 14.

说了光， 再来说影吧。 画过素描的人都知道， 开头勾勒出的轮廓是没有立体感的，然后你往恰当的位置加一些阴影，就有了立体感。 所以动物的视觉系统里存在对影子的分析处理，而且这种功能我们似乎从来没需要学习，生下来就有。“立体视觉” 是如此强烈的固化到了我们的头脑里，一旦产生了立体感，你就很难再看见平面的像素。

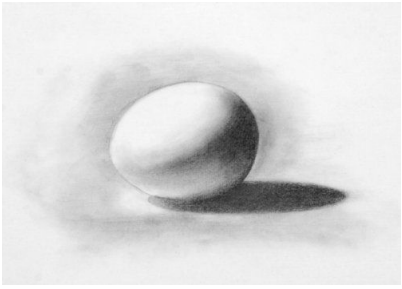


图 15.

靠着光和影的组合，人和动物能得到很多信息。比如上图，我们不但看得出这是一个立体的鸡蛋，而且能推断出鸡蛋下面是一个平面，可能是一张桌子，因为有阴影投在了上面。

神经网络知道什么是影子吗？它如何知道影子不是实际存在的物体呢？它能从影子得到有用的信息吗？

神经网络根本不知道影子是什么。早就有人发现，Tesla 基于图像识别的 Autopilot 系统会被阴影所迷惑，以为路面上的树影是一个障碍物，试图避开它，却差点撞上迎面来的车。我在很早的一篇文章已经谈过这个问题。

再来一个关于绘画的话题。学画的初期，很多人都发现画 “透视” 特别困难。所谓透视就是 “近大远小”。 本来房子的几堵墙都是长方形，是一样高的，可是你得把远的那一边画短一些， 而且相关部分的比例都要画对， 就像照片上那样， 所以墙就成了梯形的。房顶，窗户等，也全都得做相应的调整。你得这样画，看画的人才会感觉是对的，不然

就会感觉哪里不对劲，不真实。

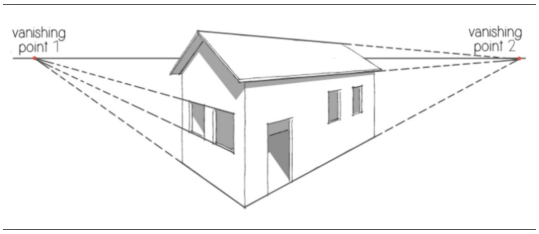


图 16.

这件事真的很难，大部分人（包括我）一辈子都没学会画透视。虽然拿起笔来量一下，我确实看到远的那一边要短一些，可是我的脑子似乎会 “自动纠错”，让我认为它们都是一样长的。所以要是光靠眼睛徒手作画，我会把那些边都画成一样长。我似乎永远学不会画画！

画透视是如此困难的事情，以至于 16 世纪的德国画家丢勒 (Albrecht Dürer) 为此设计了一种专门的设备。

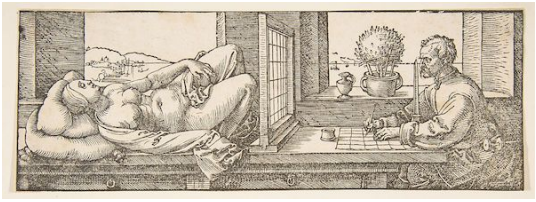


图 17.

你可能没有想到，这个使得我们学画困难的罪魁祸首，其实是人类视觉系统的一项重要功能，它帮助我们理解身边的环境。虽然眼睛看到的物体是近大远小，可是人脑会自动调整它们在你 “头脑里的长度”，所以你知道它们是一样长的。

这也许就是为什么人能从近大远小的光学成像还原出正确的 3D 模型。在你头脑中的模型里面，房子的几堵墙是一样高的，就像它们在现实中的情况一样。有了准确的 3D 模型，人才能正确地控制自己在房子周围的运动。

这种导致我们学画困难的 “3D 自动纠错” 功能，似乎固化到了每个人，每个高等动物的视觉系统里。我们并不需要学习就有这种能力，它一直都在起作用。反倒是我们要想 “关掉” 这个功能的时候，需要付出非常多的努力！

为什么人想要画出透视效果那么困难呢？因为一般人画画，都不是在画他们头上那两只眼睛看到的东西，而是在画他们的 “心之眼” (mind’s eye) 看到的东西 —— 他们头脑中的那个 3D 模型。这个 3D 模型是跟现实 “同构” 的，模型里房子的墙壁都是一样高的，他们画出来也是一样高的，所以就画错了。只有经过专业训练的画家，才有能力关闭 “心之眼”，直接画出眼睛看到的东西。

我猜想，每一种高等动物的视觉系统都有类似的机制，使得它们从光学成像 “重构” 出与现实同构的 3D 模型。缺乏 3D 建模能力的机器，是无法准确理解看到的物体的。

现在很多自动驾驶车用激光雷达构造 3D 模型，可是相对于人类视觉形成的模型，真是太粗糙了。激光雷达靠主动发射激光，产生一个扫描后的 “点云”，分辨率很低，只能形成一个粗糙的 3D 轮廓，无法识别物体，也无法理解它的结构。我们应该好好思考一下，为什么人仅靠被动接收光线就能构造出如此精密的 3D 模型，理解物体的结构，而且能精确地控制自己的动作来操作这些物体。

现在的深度学习模型都是基于像素的，没有抽象能力，不能构造 3D 拓扑模型，甚至连位置关系都分不清楚。缺乏人类视觉系统的这种 “结构理解” 能力，可能就是为什么深度学习模型需要那么多的数据，那么多的计算，才勉强能得出物体的名字。而小孩子识别物体根本不需要那么多数据和计算，看一两次就知道这东西是什么了。

人脑提取了物体的要素，所以很多信息都可以忽略了，所以人需要处理的数据量，可能比深度学习模型小很多。深度学习领域盲目地强调提高算力，制造出越来越大规模的计算芯片，GPU，TPU..... 可是大家想过人脑到底有多大计算能力吗？它可能并不需要很多计算。

从上面的各种现象，我们也许已经看明白了，人类视觉系统是很神奇的。现有的机器视觉研究并没有理解人类视觉的这些能力是怎么实现的。在接下来的续集中我们会详细的看清楚，AI 领域到底理解多少人类神经系统的构造。

9 机器与人类视觉能力的差距（2）

“神经网络” 与人类神经系统的关系是是很肤浅的。等你理解了所谓“神经网络”，就会明白它跟神经系统几乎没有一点关系。“神经网络” 只是一个误导性质的 marketing 名词，它出现的目的只是为了让外行产生不明觉厉的效果，以为它跟人类神经系统有相似之处，从而对所谓的 “人工智能” 信以为真。

其实所谓 “神经网络” 应该被叫做 “可求导编程”。说穿了，所谓 “神经网络”，“机器学习”，“深度学习”，就是利用微积分，梯度下降法，用大量数据拟合出一个函数，所以它只能做拟合函数能做的那些事情。

用了千万张图片和几个星期的计算，拟合出来的函数也不是那么可靠。人们已经发现用一些办法生成奇怪的图片，能让最先进的深度神经网络输出完全错误的结果。

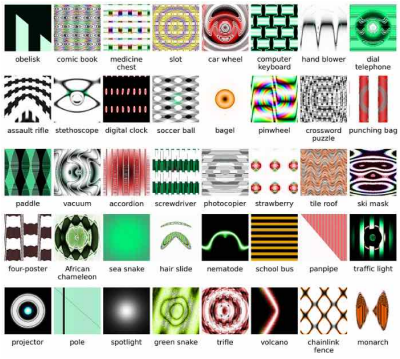


图 18.

神经网络为什么会有这种缺陷呢？因为它只是拟合了一个 “像素=>名字” 的函数。这函数碰巧能区分训练集里的图片，却不能抓住物体的结构和本质。它只是像素级别的拟合，所以这里面有很多空子可以钻。

深度神经网络经常因为一些像素，颜色，纹理匹配了物体的一部分，就认为图片上有这个物体。它无法像人类一样理解物体的结构和拓扑关系，所以才会被像素级别的肤浅假象所欺骗。

比如下面两个奇怪的图片，被认为是一个菠萝蜜和一个遥控器，仅仅因为它们中间出现了相似的纹理。

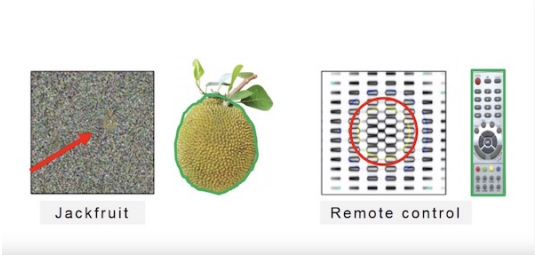


图 19.

另外，神经网络还无法区分位置关系，所以它会把一些位置错乱的图片也识别成某种物体。

神经网络为什么会犯这种错误呢？因为它的目标只是把训练集里的图片正确分类，提高“识别率”。至于怎么分类，它可以是毫无原则的，它完全不理解物体的结构。它并没有看到“叶子”，“果皮”，“方盒子”，“按钮”，它看到的只是一堆像素纹理。因为训练集里面的图片，出现了类似纹理的都被标记为“菠萝蜜”和“遥控器”，没有出现这纹理的都被标记为其它物品。所以神经网络找到了区分它们的“分界点”，认为看到这样的纹理，就一定是菠萝蜜和遥控器。

我试图从神经网络的本质，从统计学来解释这个问题。神经网络其实是拟合一个函数，试图把标签不同的样本分开。拟合出来的函数试图接近一个“真实分界线”。所谓“真实分界线”，是一个完全不会错的函数，也就是“现实”。

数据量小的时候，函数特别粗糙。数据量大了，就逐渐逼近真实分界线。但不管数据量如何大，它都不可能得到完全准确的“解析解”，不可能正好抓住“现实”。

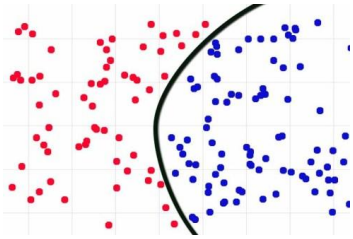


图 20.

除非现实函数特别简单，运气特别好，否则用数据拟合出来的函数，都会有很多小“缝隙”。以上的像素攻击方法，就是找到真实分界线附近，“缝隙”里面的样本，它们正好让拟合函数出现分类错误。

人的视觉系统是完全不同的，人直接就看到了事物是什么，看到了“解析解”，看到了“现实”，而没有那个用数据逼近的过程，所以除非他累得头脑发麻或者喝了酒，你几乎不可能让他判断错误。

退一步来看，图像识别所谓的“正确分类”都是人定义的。是人给了那些东西名字，是许多人一起标注了训练用的图片。所以这里所谓的“解析解”，“现实”，全都是人定义的。一定是某人看到了某个事物，他理解了它的结构和性质，然后给了它一个名字。所以别的人也可以通过理解同一个事物的结构，来知道它是什么。

神经网络不能看到事物的结构，所以它们也就难以得到精确的分类，所以机器在图像识别方面是几乎不可能超越人类的。现在所谓的 “超人类视觉” 的深度学习模型，大部分都是欺骗和愚弄大众。 使用没有普遍性的数据集， 使用不公平的准确率标准来对比，所以才显得机器好像比人还厉害了。这是一个严重的问题，在后面我会详细分析。

神经网络就像应试教育训练出来的学生，他们的目标函数是 “考高分”， 为此他们不择手段。等毕业工作遇到现实的问题，他们就傻眼了，发现自己没学会什么东西。因为他们学习的时候只是在训练自己 “从 ABCD 里区分出正确答案”。 等到现实中没有 ABCD 的时候，他们就不知道怎么办了。

深度学习训练出来的那些 “参数” 是不可解释的，因为它们存在的目的只是把数据拟合出来，把不同种类的图片分离开，而没有什么意义。AI 人士喜欢给这种 “不可解释性” 找借口，甚至有人说：“神经网络学到的数据虽然不可解释，但它却出人意料的有效。这些学习得到的模型参数，其实就是知识！”

这些模型真的那么有效吗？那为什么能够被如此离谱的图片所欺骗呢？说“那就是知识”，这说法简直荒谬至极，严重玷污了“知识”这个词的意义。这些“学习”得到的参数根本就不是本质的东西，不是知识，真的就是一堆毫无道理可言的数字，只为了降低“误差”，能够把特征空间的图片区分开来，所以神经网络才能被这样钻空子。

说这些参数是知识，就像在说考试猜答案的技巧是知识一样可笑。“另外几套题的第十题都是 B，所以这套题的第十题也选 B”..... 深度学习拟合函数，就像拿历年高考题和它们的答案来拟合函数一样， 想要不上课， 不理解科目知识就做出答案来。 有些时候它确实可以蒙对答案，但遇到前所未见的题目，或者题目被换了一下顺序，就傻眼了。

人为什么可以不受这种欺骗呢？因为人提取了高级的拓扑结构，不是瞎蒙的，所以人的判断不受像素的影响。 因为提取了结构信息， 人的观察是具有可解释性的。 如果你问一个小孩，为什么你说这是一只猫而不是一只狗呢？她会告诉你：“因为它的耳朵是这样的，它的牙是那样的，它走路的姿势是那样的，它常常磨爪子，它用舌头舔自己.....”

做个实验好了， 你可以问问你家孩子这是猫还是狗。 如果是猫， 为什么他们认为这是一只猫而不是一只狗？

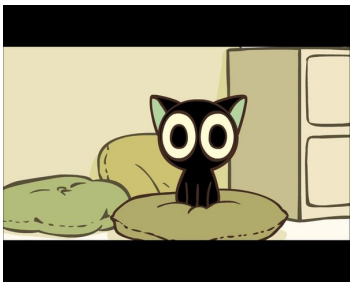


图 21.

神经网络看到一堆像素，很多层处理之后也不知道是什么结构，分不清 “眼睛”，“耳朵” 和 “嘴”， 更不要说 “走路” 之类的动态概念了，所以它也就无法告诉你它认为这是猫的原因了。拟合的函数碰巧把这归成了猫，如果你要追究原因，很可能是肤浅的：图片上有一块像素匹配了图片库里某只猫的毛色纹理。

有一些研究者把深度神经网络的各层参数拆出来，找到它们对应的图片中的像素和纹理，以此来证明神经网络里的参数是有意义的。咋一看好像有点道理，原来 “学习” 就能得到这么多好像设计过的滤镜啊！ 可是仔细一看， 里面其实没有多少有意义的内容，因为它们学到的参数只是能把那些图片类别分离开。

所以人的视觉系统很可能是跟深度神经网络原理完全不同的， 或者只有最低级的部分有相似之处。

为什么 AI 人士总是认为视觉系统的高级功能都能通过 “学习” 得到呢？非常可能的事情是，人和动物视觉系统的 “结构理解”， “3D建模” 功能不是学来的，而是早就固化在基因里了。想一想你生下来之后，有任何时候看到世界是平面的，毫无关联的像素吗？

所以我觉得， 人和动物生下来就跟现有的机器不一样，结构理解所需的硬件在胚胎里就已经有了，只等发育和激活。人是有学习能力，可是人的学习是建立在结构理解之上，而不是无结构的像素。另外人的 “学习” 很可能处于比较高的层面，而不是神经元那么 “底层” 的。人的神经系统里面并没有机器学习那种 back-propagation.

纵使你有再多的数据，再多的计算力，你能超越为期几十亿年的，地球规模的自然进化和选择吗？与其自己去 “训练” 或者 “学习”， 不如直接从人身上抄过来！但问题是，我们真的知道人的视觉系统是如何工作的吗？

神经科学家们其实并没有完全搞明白人类视觉系统是如何工作的。 就像所有的生物学领域一样， 人们的理解仍然是很粗浅的。 神经网络与人类视觉系统的关系是肤浅的。每当你质疑神经网络与人类视觉系统的关系， AI 研究者就会抬出 Hubel & Wiesel 在 1959 年拿猫做的那个实验：“有人已经证明了人类视觉系统就是那样工作的！” 如此的自信，不容置疑的样子。

我问你啊，如果我们在 1959 年就已经知道人类视觉系统的工作原理细节，为什么现在还各种模型改来改去，训练来训练去呢？直接模仿过来不就行了？所以这些人的说法是自相矛盾的。

你想过没有，为什么到了 2019 年，AI 人士还拿一个 60 年前的实验来说明问题？这 60 年来就没有新的发现了吗？而且从 H&W 的实验你可以看出来，它只说明了猫的视觉神经有什么样的底层功能（能够做 “线检测”），却没有说那就是全部的构造，没说上层的功能都是那样够构造的。

H&W 的实验只发现了最底层的 “线检测”， 却没有揭示这些底层神经元的信号到了上层是如何组合在一起的。“线检测” 是图像处理的基础操作。一个能够识别拓扑结构的动物视觉系统，理所当然应该能做 “线检测”， 但它应该不止有这种低级功能。

视觉系统应该还有更高级的结构，H&W 的实验并没能回答这个问题，它仍然是一个黑盒子。AI 研究者们却拿着 H&W 的结果大做文章，自信满满的声称已经破解了动物视觉系统的一切奥秘。

那些说 “我们已经完全搞明白了人类视觉是如何工作” 的 AI 人士，应该来看看这个 2005 年的分析 Herman grid 幻觉现象的幻灯片。这些研究来自 Schiller Lab，MIT 的脑科学和认知科学实验室。通过一系列对 Herman grid 幻觉图案的改动实验，他们发现长久以来（从 1960 年代开始）对产生这种现象的理解是错误的：那些暗点不是来自视网膜的 “边沿强化” 功能。他们猜想，这是来自大脑的 V1 视觉皮层的 S1 “方向选择” 细胞。接着，另一篇 2008 年的 paper 又说，Schiller 的结果是不对的，这种幻觉跟那些线条是直的有关系，因为你如果把那些白线弄弯，幻觉就消失了。然后他们提出了他们自己的，新的 “猜想”。

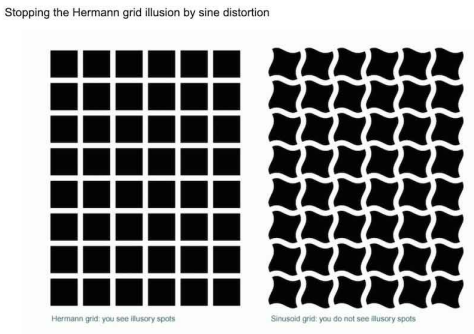


图 22.

从这种研究的方式我们可以看出，即使是 MIT 这样高级的研究所，对视觉系统的研究还处于 “猜” 的阶段，把人脑作为黑盒子，拿一些图片来做 “行为” 级别的实验。他们并没有完全破解视觉系统，看到它的 “线路” 和 “算法” 具体如何工作，而是给它一些输入，测试它的输出。这就是 “黑盒子” 实验法。以至于很多关于人类视觉的理论都不是切实而确定的，很可能是错误的猜想。

脑科学发展到今天也还是如此， AI 领域相对于脑科学的研究方式， 又要低一个级别。2019 年了，仍然抬出神经科学家 1959 年的结果来说事。闭门造车，对人家的最新成果一点都不关心。 现在的深度神经网络模型基本是瞎蒙出来的。 把一堆像素操作叠在一起，然后对大量数据进行 “训练”， 以为这样就能得到所有的视觉功能。

动物视觉系统里面真有 “反向传导” (back-propagation) 这东西吗？H&W 的实验里面并没有发现 back-propagation. 实际上神经科学家们至今也没有发现神经系统里面有 back-propagation, 因为神经元的信号传递机制不能进行 “反向” 的通信。很多神经科学家的结论是，人脑里面进行 back-propagation 不大可能。

所以神经网络的各种做法恐怕没有受到 H&W 实验的多大启发。只是靠这么一个肤浅的相似之处来显得自己接近了 “人类神经系统”。 现在的所谓 “神经网络”， 其实只是一个普通的数学函数的表达式，里面唯一起作用的东西其实是微积分，所谓 back-propagation, 就是微积分的求导操作。神经网络的 “训练”， 就是反复求导数，用梯度下降方法进行误差最小化，拟合一个函数。这一切都跟神经元的工作原理没什么关系，完全就是数学。

为了消除无知带来的困惑，你可以像我一样，自己去了解一下人类神经系统的工作原理。我推荐你看看这个叫《Interactive Biology》的 YouTube 视频系列。你可以从中轻松地理解人类神经系统一些细节：神经元的工作原理，视觉系统的原理，眼睛，视网膜的结构，听觉系统的工作原理，等等。 神经学家们对此研究到了如此细节的地步，神经传导信息过程的每一个细节都展示了出来。

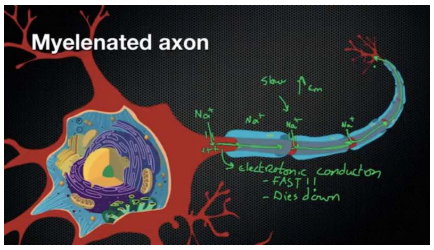


图 23.

AI 领域真的理解人脑如何工作吗？你可以参考一下这个演讲："Can the brain do back-propagation?" （人脑能做 back-propagation 吗？）。演讲人是深度学习的鼻祖级人物 Geoffrey Hinton. 他和其它两位研究者（Yoshua Bengio 和 Yann LeCun），因为对深度学习做出的贡献，获得了 2018 年的图灵奖。演讲一开头 Hinton 说，神经科学家们说人脑做 back-propagation 是不可能的，然后他开始证明这是可能的，依据神经元的工作原理，back-propagation 如何能用人脑神经元来实现。

是的，如果你有能力让人脑按你的 “算法” 工作的话，神经元组成的系统也许真能做 back-propagation, 可是人脑是你设计的吗？很可惜我们无法改变人脑，而只能去 “发现” 它到底是如何工作。这不是人脑 “能不能” 的问题，而是 “做不做” 的问题。研究人脑是一个科学发现工作，而不是一个工程设计工作。

看了这个演讲，我觉得 AI 人士已经进入了一种 “上了天” 的状态。他们坚定的认为自己的模型（所谓的 “神经网络”）就是终极答案，甚至试图把人脑也塞进这个模型，设想人脑神经元如何能实现他们所谓的 “神经网络”。可是他们没有发现，人脑的方式也许比他们的做法巧妙很多，根本跟他们的 “神经网络” 不一样。

从这个视频我们也可以看出，神经科学界并不支持 AI 领域的说法。AI 领域是自己在那里瞎猜。视频下面有一条评论我很欣赏，他用讽刺的口气说："Geoff Hinton 确切地知道人脑是如何工作的，因为这是他第 52 次发现人脑工作的新方式。"

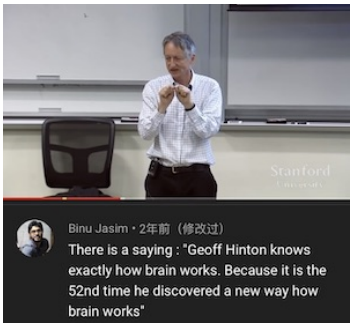


图 24.

AI 人士似乎总是有一种不切实际的“信仰”或者“信念”，他们坚信机器一定可以具有人类一样的智能，总有一天能够在所有方面战胜人类。总是显示出一副“人类没什么了不起”的心态，张口闭口拿“人类”说事，好像他们自己是另外一个物种，已经知道人类的一切能力，有资格评判所有人的智力似的。

我不知道是什么导致了这种 “AI 宗教”。有句话说得好：“我所有的自负都来自我的自卑，所有的英雄气概都来自于我内心的软弱，所有的振振有词都因为心中满是怀疑。” 似乎是某种隐藏很深的自卑和怨恨，导致了他们如此的坚定和自负。一定要搞出个超越所有人的机器才善罢甘休，却没发现人类智能的博大精深已经从日常生活的各种不起眼的小事透露出来。

他们似乎看不到世界上有各种各样，五花八门的人类活动，每一种都显示出奇迹般的智能。连端茶倒水这么简单的事情，都包含了机器望尘莫及的智能，更不要说各种体育运动，音乐演奏，各种研究和创造活动了。就连比人类 “低级” 一点的动物，各种宠物，家畜家禽，飞鸟走兽，甚至昆虫，全都显示出足以让人敬畏的智能。他们对所有这些奇迹般的事物视而不见，不是去欣赏他们的精巧设计和卓越表现，而是坐井观天，念叨着 “机器一定会超越人类”。

他们似乎已经像科幻电影似的把机器当成了一个物种，像是保护 “弱势群体” 一样，要维护机器的 “权益” 和 “尊严”。 他们不允许其他人质疑这些机器，不允许你说它们恐怕没法实现人类一样的智能。 总之机器在他们心理已经不再是工具， 而是活的生命， 甚至是比人还高级的生命。

对此你可以参考另一个 Geoffrey Hinton 的采访视频 (A Fireside Chat with Turing Award Winner Geoffrey Hinton, Pioneer of Deep Learning), 录制于 Google 开发者大会（Google I/O ‘19）。

从这个视频里面我看到了许多 AI 人士盲目信仰和各种没有根据的说法的来源，因为这些说法全都集中而强烈的体现在了 Hinton 的谈话中。他如此的坚信一些没有根据的说法，不容置疑地把它们像真理一样说出来，却没有任何证据。有时候主持人都不得不采用了有点怀疑的语气。

Hinton 在采访中有以下说法：

1. “神经网络被设计为像人脑的工作原理。”
2. “等神经网络能够跟人对话，我们就能用它来进行教育工作了。”
3. “神经网络终究会在所有事情上战胜人类。”
4. “我们不都是神经网络吗？”（先后强调了两次）
5. “..... 所以神经网络能够实现人类智能的一切功能。这包括感情，意识等。”
6. “人们曾经认为生命是一种特殊的力量，现在生物学解释了生命的一切。人们现在仍然认为意识是特殊的，可是神经网络将会说明，意识并没有什么特别。”

图 25.

他的这些说法都是不准确，不科学，没有根据的。

我发现每当主持人用稍微怀疑的语气问：“这真的可以实现吗？” Hinton 就会回答：“当然能。我们不都是神经网络吗？” 这里有一个严重的问题，那就是他所谓的 “神经网络”，其实并不是人脑里面的神经元连成的网络。AI 领域的 “神经网络” 只是他们自己的数学模型，是他们自己给它起名叫 “神经网络” 而已。所以他的这种 “证明” 其实是在玩文字游戏：“因为我们都是神经网络，所以神经网络能够实现一切人类智能，感情，甚至意识本身！”

前面的 “神经网络” 和后面的 “神经网络” 完全是两回事。我们是 “神经网络” 吗？我们的脑子里是有神经元，神经元貌似连成了一个网络，可是它的结构却跟 AI 领域所谓的 “神经网络” 是两回事，工作原理也非常不一样。Hinton 面对问题作出这样的回答，是非常不科学，不负责任的。

最后关于生命，感情和意识的说法，我也很不认同。虽然生物学解释了生命体的各种构造和原理，可是人们为什么仍然没能从无生命的物质制造出有生命的事物呢？虽然人们懂得那么多生物学，生物化学，有机化学，甚至能合成出各种蛋白质，可是为什么

没能把这些东西组装在一起，让它 “活” 起来呢？这就像你能造出一些机器零件，可是组装起来之后， 发现这机器不转。 你不觉得是因为少了点什么吗？ 生物学发展了这么久，我们连一个最简单的，可以说是 “活” 的东西都没造出来过，你还能说 “生命没什么特别的 ” 吗？

这说明生物学家们虽然知道生命体的一些工作原理，却没有从根本上搞明白生命到底是什么。也就是说人们解决了一部分 "how" 问题（生命体如何工作），却不理解 "what" 和 "why"（生命是什么，为什么会出现生命）。

实际上生物学对生命体如何工作（how）的理解都还远远不够彻底， 这就是为什么我们还有那么多病无法医治， 甚至连一些小毛病都无法准确的根治，一直拖着， 只是不会马上致命而已。“生命是什么” 的 what 问题仍然是一个未解之谜，而不像 Hinton 说的，全都搞明白了，没什么特别的。

也许生命就是一种特别的东西呢？也许只有从有生命的事物， 才能产生有生命的事物呢？也许生命就是从外星球来的，也许就是由某种更高级的智慧设计出来的呢？这些都是有可能的。 真正的科学家应该保持开放的心态， 不应该有类似“人定胜天”这样的信仰。我们的一切结论都应该有证据，如果没有我们就不应该说“一定”或者“必然”，说得好像所有秘密全都解开了一样。

对于智能和意识，我也是一样的态度。在我们没有从普通的物质制造出真正的智能和意识之前，不应该妄言理解了关于它们的一切。生命，智能和意识，比有些人想象的要奇妙得多。想要“人造”出这些东西，比 AI 人士的说法要困难许多。

有心人仔细观察一下身边的小孩子，小动物，甚至观察一下自己，就会发现它们的 “设计” 是如此的精巧， 简直不像是随机进化出来的， 而是由某个伟大的设计者创造的。46 亿年的时间，真的够进化和自然选择出这样聪明的事物吗？

别误会了， 我是不信宗教的。 我觉得宗教的圣经都是小人书， 都是某些人吓编的。可是如果你坚定的相信人类和动物的这些精巧的结构都是 “进化” 来的，你坚定的相信它们不是什么更高级的智慧创造出来的，那不也是另外一种宗教吗？你没有证据。没有证据的东西都只是猜想，而不能坚信。

好像扯远了……

总之，深度学习的鼻祖级人物说出这样多信念性质的，没有根据的话，由此可见这个领域有多么混沌。另外你还可以从他的谈话中看出，他所谓的 "AI" 都是各种相对容易的识别问题（语音识别，图像识别）。他并没有看清楚机器要想达成 “理解” 有多困难。而 “识别” 与 “理解” 的区别，就是我的这篇文章想澄清的问题。

图 26.

设计神经网络的 “算法工程师”， “数据科学家”， 他们工作性质其实很像 “炼丹师” (alchemist). 拿个模型这改改那改改， 拿海量的图片来训练，“准确率” 提高了， 就发 paper. 至于为什么效果会好一些，其中揭示了什么原理，模型里的某个节点是用来达到什么效果的， 如果没有它会不会其实也行？ 不知道， 不理解。 甚至很多 paper 里的结果无法被别的研究者复现，存在作假的可能性。

我很怀疑这样的研究方式能够带来什么质的突破， 这不是科学的方法。 如果你跟我一样，把神经网络看成是用 “可求导编程语言” 写出来的代码，那么现在这种设计模型的方法就很像 “一百万只猴子敲键盘”， 总有一只能敲出 “Hello World！”

许多数学家和统计学家都不认同 AI 领域的研究方式，对里面的很多做法表示不解和怀疑。为此斯坦福大学的统计学系还专门开了一堂课 Stats 385，专门讨论这个问题。课堂上请来了一些老一辈的数学家， 一起来分析深度学习模型里面的各种操作是用来达到什么目的。有一些操作很容易理解，可是另外一些没人知道是怎么回事，这些数学家都看不明白，连设计这些模型的炼丹师们自己都不明白。

所以你也许看到了，AI 研究者并没能理解人类视觉系统的工作原理，许多的机器视觉研究都是在瞎猜。在接下来的续集中，我们会看到他们所谓的“超人类识别率”是如何来的。

10 机器与人类视觉能力的差距（3）

我发现神经网络在测试数据的可靠性，准确率的计算方法上，都有严重的问题。

神经网络进行图像识别，所谓 “准确率” 并不是通过实际数据测出来的，而是早就存在那里的，专用的测试数据。比如 ImageNet 里面有 120 万张图片，是从 Flickr 等照片网站下载过来的。反反复复都是那些，所以实际的准确率和识别效果值得怀疑。数据全都是网络上的照片， 但网络上数据肯定是不全面的， 拍照的角度和光线都无法概括现实的多样性。 而且不管是训练还是测试的数据， 他们选择的都是在理想环境下的照片， 没有考虑各种自然现象：反光，折射，阴影等。

比如下图就是图像识别常用的 ImageNet 和其它几个数据集的一小部分。你可以看到它们几乎全都是光线充足情况下拍的照片， 训练和测试用的都是这样的照片， 所以遇到现实的场景，光线不充足或者有阴影，准确率很可能就没有 paper 上那么高了。

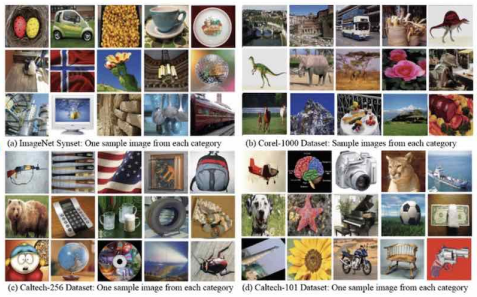
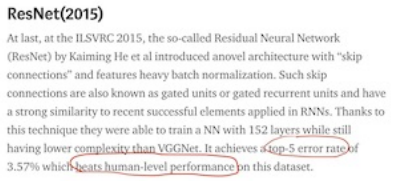


图 27.

如此衡量“准确率”，有点像你做个编译器，却只针对很小一个 benchmark 进行优化跑分。一旦遇到实际的代码，别人可能就发现性能不行。但神经网络训练需要的硬件等条件比较昂贵，一般人可能也很少有机会进行完整的模型训练和实际的测试，所以大家只有任凭业内人士说“超人类准确率”，却无法验证它的实际效果。

不但测试数据的 “通用性” 值得怀疑，所谓 “准确率” 的计算标准也来的蹊跷。AI 领域向公众宣扬神经网络准确率的时候，总喜欢暗地里使用所谓 “top-5 准确率”，也就是说每张图片给 5 次机会分类，只要其中一个对了就算正确，然后计算准确率。依据 top-5 准确率，他们得出的结论是，某些神经网络模型识别图像的准确率已经 “超越了人类”。


ResNet(2015)

At last, at the ILSVRC 2015, the so-called Residual Neural Network (ResNet) by Kaiming He et al introduced anovel architecture with “skip connections” and features heavy batch normalization. Such skip connections are also known as gated units or gated recurrent units and have a strong similarity to recent successful elements applied in RNNs. Thanks to this technique they were able to train a NN with 152 layers while still having lower complexity than VGGNet. It achieves a top-5 error rate of 3.57% which beats human-level performance on this dataset.

图 28.

如果他们提到 "top-5" 还算好的了，大部分时候他们只说 “准确率”，而不提 "top-5" 几个字。在跟人比较的时候，总是说 “超越了人类”，而绝口不提 "top-5"，不解释是按照什么标准。我为什么对 top-5 有如此强烈的异议呢？现在我来解释一下。

具体一点，"top-5" 是什么意思呢？也就是说对于一张图片，你可以给出 5 个可能的分类，只要其中一个对了就算分类正确。

比如图片上本来是汽车，我看到图片，说：

1. “那是苹果？”
2. “哦不对，是杯子？”
3. “还是不对，那是马？”
4. “还是不对，所以是手机？”
5. “居然还是不对，那我最后猜它是汽车！”

五次机会，我说出 5 个风马不及的词，其中一个对了，所以算我分类正确。荒谬吧？这样继续，给很多图片分类，然后统计你的“正确率”。

为什么要给 5 次机会呢？ImageNet 比赛（ILSVRC）对两种不同的比赛给出了两种不大一样的说法。一种说是为了让机器可以识别出图片上的多个物体，而不因为其中某个识别出的物体不是正确标签（ground truth）而被算作错误。另外一种说是为了避免输出意义相同的近义词，却不能完全匹配标签而被算作错误。

两个说法的理由不同，但数学定义基本是一样的。总之就是有五次机会，只要对了一个就算你对。

II: Object localization

The data for the classification and localization tasks will remain unchanged from ILSVRC 2012 . The validation and test data will consist of 150,000 photographs, collected from flickr and other search engines, hand labeled with the presence or absence of 1000 object categories. The 1000 object categories contain both internal nodes and leaf nodes of ImageNet, but do not overlap with each other. A random subset of 50,000 of the images with labels will be released as validation data included in the development kit along with a list of the 1000 categories. The remaining images will be used for evaluation and will be released without labels at test time. The training data, the subset of ImageNet containing the 1000 categories and 1.2 million images, will be packaged for easy downloading. The validation and test data for this competition are not contained in the ImageNet training data.

In this task, given an image an algorithm will produce 5 class labels $c_i, i = 1, \ldots 5$ in decreasing order of confidence and 5 bounding boxes $b_i, i = 1, \ldots 5$, one for each class label. The quality of a localization labeling will be evaluated based on the label that best matches the ground truth label for the image and also the bounding box that overlaps with the ground truth. The idea is to allow an algorithm to identify multiple objects in an image and not be penalized if one of the objects identified was in fact present, but not included in the ground truth.

The ground truth labels for the image are $C_k, k = 1, \ldots n$ with n class labels. For each ground truth class label C_k, the ground truth bounding boxes are $B_{km}, m = 1 \ldots M_k$, where M_k is the number of instances of the k^{th} object in the current image.

Let $d(c_i, C_k) = 0$ if $c_i = C_k$ and 1 otherwise. Let $f(b_i, B_k) = 0$ if b_i and B_k have more than 50% overlap, and 1 otherwise. The error of the algorithm on an individual image will be computed using:

$$e = \frac{1}{n} \cdot \sum_k min_i min_m max\{d(c_i, C_k), f(b_i, B_{km})\}$$

The winner of the object localization challenge will be the team which achieves the minimum average error across all test images.

For each image, algorithms will produce a list of at most 5 scene categories in descending order of confidence. The quality of a labeling will be evaluated based on the label that best matches the ground truth label for the image. The idea is to allow an algorithm to identify multiple scene categories in an image given that many environments have multi-labels (e.g. a bar can also be a restaurant) and that humans often describe a place using different words (e.g. forest path, forest, woods).

For each image, an algorithm will produce 5 labels $l_j, j = 1, \ldots, 5$. The ground truth labels for the image are $g_k, k = 1, \ldots, n$ with n classes of scenes labeled. The error of the algorithm for that image would be

$$e = \frac{1}{n} \cdot \sum_k \min_j d(l_j, g_k).$$

$d(x, y) = 0$ if $x = y$ and 1 otherwise. The overall error score for an algorithm is the average error over all test images. Note that for this version of the competition, n=1, that is, one ground truth label per image.

图 29.

看似合理？然而这却是模糊而错误的标准。这使得神经网络可以给出像上面那样风马不及的 5 个标签（苹果，杯子，马，手机，汽车），其中前四个都不是图片上的物体，却仍然被判为正确。

你可能觉得我的例子太夸张了，但是准确率计算标准不应该含有这样的漏洞。只要标准有漏洞，肯定会有错误的情况会被放过。现在我们来看一个实际点的例子。

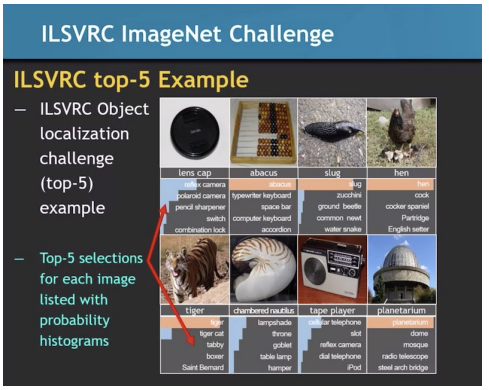


图 30.

上图是一个 Coursera 的机器学习课程给出的 top-5 实际输出结果的例子。你可以从中发现，纵然有一些 top-5 输出标签是近义词，可是也有很多并不是近义词，而是根本错误的标签。比如“算盘”图片的 top-5 里面包含了computer keyboard（电脑键盘）和 accordion（手风琴）。“老虎”图片的 top-5 里面包含了两种狗的品种名字（boxer，Saint Bernard）。

另外你还可以看到，测试图片是经过精心挑选和裁剪的，里面很少有多于一个物体。所以第一种说法，“可能输出某个图片上存在的物体但却不是正确答案”，恐怕是很少见的。

所以 ILSVRC 对使用 top-5 给出的两个理由是站不住脚的。它想要解决的问题并不是那么突出地存在，但是它却开了一道后门，可能放过很多的错误情况。比如上面的 “算盘” 图片，如果排名第一的不是 abacus，而是 computer keyboard（电脑键盘）或者 accordion（手风琴），只要 abacus 出现在 top-5 列表里，这个图也算识别正确。所以 top-5 根本就是错误的标准。

其实要解决图片上有多个物体的问题，或者输出是近义词的问题，都有更好的办法，而不会让错误的结果被算成正确的。每一个学过基础数据结构和算法的本科生都应该能想出更好的解决方案。比如你可以用一个近义词词典，只要输出的标签和“正确标签”是近义词就算正确。对于有多个物体的图片，你可以在标注时给它多个标签，算法给出的标签如果在这个“正确标签集合”里面就算正确。

但 ILSVRC 并没有采用这些解决方案，而是采用了 top-5. 这么基础而重要的问题，AI 业界的解决方案如此幼稚，却被全世界研究者广泛接受。你们不觉得蹊跷吗？我觉得他们有自己的目的：top-5 使得神经网络的准确率显得很高，只有使用这个标准，神经网络才会看起来 “超越了人类”。

Top-5 准确率总是比 top-1 高很多。高多少呢？比如 ResNet-152 的 top-1 错误率是 19.38%, 而 top-5 错误率却只有 4.49%. Top-1 准确率只能算 “勉强能用”，换成 top-5 之后，忽然就可以宣称 “超越人类” 了，因为据说人类的 top-5 错误率大概是 5.1%.

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

图 31.

可能很多人还没意识到，top-5 比较方法对人是不公平的。图片上要是人见过的物体，几乎总是一次就能做对，根本不需要 5 次机会。使用 “top-5 准确率”，就像考试的时候给差等生和优等生各自 5 次机会来做对题目。当然，这样你就分不清谁是差等生，谁是优等生了。“top-5 准确率” 大大的模糊了好与坏之间的界线，最后看起来都差不多了，甚至差等生显得比优等生还要好。

具体一点。假设一个人识别那些图片的时候，他的 top-5 错误率是 5.1% （就像他们给出的数字那样），那么他的 top-1 错误率大概也是 5.1%. 因为人要是一次机会做不对，那他可能根本就没见过图片上的物体。如果他一次做不对，你给他 5 次机会，他也做不对，因为他根本就不知道那东西叫什么名字。

现在某个神经网络（ResNet-152）的 top-5 错误率是 4.49%, 它的 top-1 错误率是 19.38%. 你却只根据 top-5 得出结论，说神经网络超越了人类。是不是很荒谬？

退一万步讲，就算你可以用 top-5, 像这种 4.49% 与 5.1% 的差别，只相差 0.61%, 也应该是忽略不计的。因为实验都是有误差，有随机性的，根据测试数据的不同也有差

异，像这样的实验，1% 以内的差别根本不能说明问题。如果你仔细观察各个文献列出来识别率，就会发现它们列出的数字都不大一样。同样的模型，准确率差距可以有 3% 以上。但他们拿神经网络跟人比，却总是拿神经网络最好的那个数，跟人死扣那百分之零点几的 “优势”，然后欢天喜地宣称已经 “超人类” 了。

而且他们真的拿人做过公平的实验吗？为什么从来没有发布过 “神经网络 vs 人类 top-1 对比结果” 呢？5.1% 的 “人类 top-5 准确率” 数字是哪里来的呢？哪些人参加了这个测试，他们都是什么人？我唯一看到对人类表现的描述，是在 Andrej Karpathy 的主页上。他拿 ImageNet 测试了自己的识别准确率，发现好多东西根本没见过，不认识，所以他又看 ImageNet 的图片 “训练” 自己，再次进行测试，结果准确率大大提高。

就那么一个人得出的 “准确率”，就能代表全人类吗？而且你们知道 Andrej Karpathy 是谁吧。他是李飞飞的学生，目前是 Tesla 的 AI 主管，而李飞飞是 ImageNet 的发起者和创造者。让一个 “内幕人士” 拿自己来测试，这不像是公正和科学的实验方法。你见过有医学家，心理学家拿自己做个实验，就发表结果的吗？第一，人数太少，至少应该有几十个智商正常的人来做这个，然后数据平均一下吧？第二，这个人是个内幕人士，他的表现恐怕不具有客观性。

别误会了，我并不否认 Andrej Karpathy 是个很聪明，说话挺耿直的人。我很欣赏他讲的斯坦福 cs231n 课程，通过他的讲述我第一次明白了神经网络到底是什么，明白了 back-propagation 到底如何工作。我也感谢李飞飞准备了这门课，并且把它无私地放在网上。但是这么大一个领域，这么多人，要提出 “超越了人类视觉” 这么大一个口号，居然只有研究者自己一个人挺身而出做了实验，你不觉得这有点不负责任吗？

AI 领域对神经网络训练进行各种优化，甚至专门针对 top-5 进行优化，把机器的每一点性能每一点精度都想榨干了去，对于如何让人准确显示自己的识别能力，却漫不经心，没有组织过可靠的实验，准确率数字都不知道是怎么来的。对比一下生物，神经科学，医学，这些领域是如何拿人做实验，如何向大家汇报结果，AI 领域的做法像是科学的吗？

这就是 “AI 图像识别超越人类” 这种说法来的来源。AI 业界所谓 “超人类的识别率”，“90+% 的准确率”，全都是用 “top-5 准确率” 为标准的，而且用来比较的人类识别率的数字没有可靠的来源。等你用 “top-1 准确率” 或者更加公平的标准，使用客观公正抽选的人类实验者的时候，恐怕就会发现机器的准确率远远不如人类。

我们来看看 top-1 准确率吧。业界最先进的模型之一 ResNet-152 的 top-1 错误率是 19.38%. 2017 年的 ImageNet 分类冠军 SENet-154, top-1 错误率是 18.68%. 当然这也没有考虑过任何实际的光线，阴影和扭曲问题，只是拿标准的，理想情况的 ImageNet “测试图片” 来进行。遇到实际的情况，准确率肯定会更低。

Table 1. Single crop validation error on ImageNet-1k (center 224x224 crop from resized image with shorter side = 256). The SENet-154 is one of our superior models used in ILSVRC 2017 Image Classification Challenge where we won the 1st place (Team name: WMW).

Model	Top-1	Top-5	Size	Caffe Model	Caffe Model
SE-BN-Inception	23.62	7.04	46 M	GoogleDrive	BaiduYun
SE-ResNet-50	22.37	6.36	107 M	GoogleDrive	BaiduYun
SE-ResNet-101	21.75	5.72	189 M	GoogleDrive	BaiduYun
SE-ResNet-152	21.34	5.54	256 M	GoogleDrive	BaiduYun
SE-ResNeXt-50 (32 x 4d)	20.97	5.54	105 M	GoogleDrive	BaiduYun
SE-ResNeXt-101 (32 x 4d)	19.81	4.96	187 M	GoogleDrive	BaiduYun
SENet-154	18.68	4.47	440 M	GoogleDrive	BaiduYun

图 32.

神经网络要想提高 top-1 准确率已经非常困难了，都在 80% 左右徘徊。有些算法工程师告诉我，识别率好像已经到了瓶颈，扩大模型的规模才能提高一点点。可是更大的模型具有更多的参数，也就需要更大规模的计算能力来训练。比如 SENet-154 尺寸是 ResNet-152 的 1.7 倍，ResNet-152 尺寸又是 ResNet-50 的 2.4 倍，top-1 准确率才提高一点点。

我还有一个有趣的发现。如果你算一下 ResNet-50 和 ResNet-152 的差距，就会发现 ResNet-152 虽然模型大小是 ResNet-50 的 2.4 倍，它的 top-1 错误率绝对值却只降低了 1.03%. 从 22.37% 降低到 21.34%，相对降低了 (22.37-21.24)/22.37 = 4.6%, 很少。可是如果你看它的 top-5 错误率，就会觉得它好了不少，因为它从 6.36% 降低到了 5.54%，虽然绝对值只少了 0.82%, 比 top-1 错误率的改进还小，可是相对值却降低了 (6.36-5.54)/6.36 = 12.9%, 就显得改进了挺多。

这也许就是为什么 AI 业界用 top-5 的第二个原因。因为它的错误率基数很小，所以你减小一点点，相对的 “改进” 就显得很多了。然后你看历年对 top-5 的改进，就觉得神经网络识别率取得了长足的进步！

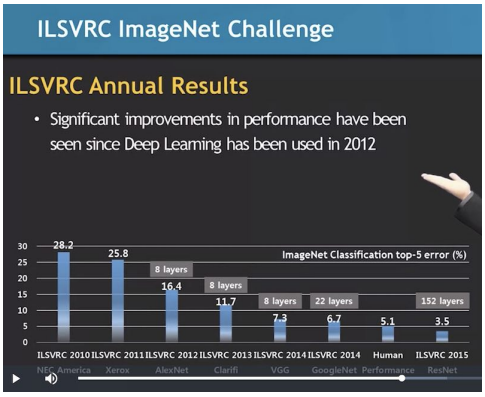


图 33.

而如果你看 top-1 准确率，就会觉得几乎没有变化。模型虽然大了几倍，计算量大了那么多，top-1 准确率却几乎没有变。所以神经网络的 top-1 准确率似乎确实到了一个瓶颈，如果没有本质的突破，恐怕再大的模型也难以超越人类。

准确率不够高，不如人类其实问题不大，只要你承认它的局限性，把它用到能用的地方就行了。可是最严重的问题是人的诚信，AI 人士总是夸大图像识别的效果，把它推向超出自己能力的应用。

AI 业界从来没有向公众说清楚他们所谓的 “超人类识别率” 是基于什么标准，反而在各种媒体宣称 “AI 已经超越了人类视觉”。 这完全是在欺骗和误导公众。上面 Geoffrey Hinton 的采访视频中，主持人也提到 “神经网络视觉超越了人类”， 这位深度学习的先驱者对此没有任何说明，而是欣然接受，继续自豪地夸夸其谈。

你可以给自动驾驶车 5 次机会来判断前面出现的是什么物体吗？你有几条命可以给它试验呢？Tesla 的 Autopilot 系统可能 top-5 正确率很高吧：“那是个白板..... 哦不对，那是辆卡车！” “那是块面包..... 哦不对，那是高速公路的隔离带！”

我不是开玩笑，上面的 “卡车” 和 “隔离带”， 它们指向的是 Tesla Autopilot 引起的两次致命车祸。第一次车祸，Autopilot 把卡车识别为白板，直接从侧面撞上去，导致车主立即死亡。另一次，它开出车道，没能识别出高速公路中间的隔离带， 完全没有减速，反而加速撞上去，导致车主死亡，并且着火爆炸。

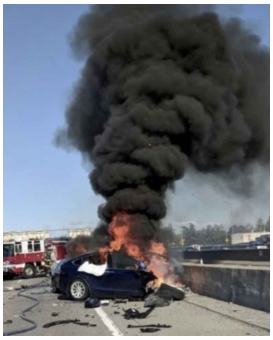


图 34.

神经网络能把卡车识别为白板还算 “top-5 分类正确”， Autopilot 根本没有视觉理解能力，这就是为什么会引起这样可怕的事故。

图 35.

你可以在 Wikipedia (Tesla_Autopilot#Incidents) 上看到一个 Autopilot 导致的事故列表。

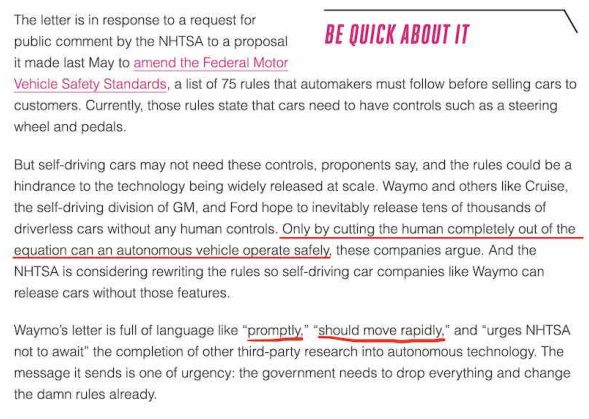
出了挺多人命，可是“自动驾驶”的研究仍然在混沌中进行。2018 年 3 月，Uber 的自动驾驶车在亚利桑那州撞死一名推自行车过马路的女性。 事故发生时的车载录像已经被公布到了网上。

报告显示，Uber 的自动驾驶系统在出事前 6 秒钟检测到了这位女士，起初把她分类为 “不明物体”， 然后分类为 “汽车”， 最后分类为 “自行车”， 完全没有刹车，以每小时 40 英里的速度直接撞了上去……

在此之前， Uber 被加州政府吊销了自动驾驶实验执照， 后来他们转向了亚利桑那州，因为亚利桑那州长热情地给放宽政策，“拥抱高科技创新”。结果呢，搞出人命来了。美国人看到 Uber 自动车撞死人， 都在评论说， 要实验自动驾驶车就去亚利桑那州吧，因为那里的人命不值钱，撞死不用负责！

据 2018 年 12 月消息，Uber 想要重新开始自动驾驶实验，这次是在宾夕法尼亚州的匹兹堡。 他们想要在匹兹堡的闹市区进行自动驾驶实验， 因为那里有狭窄的街道， 列车铁轨，许多的行人…… 我觉得要是他们真去那里实验，可能有更好的戏看了。

自动驾驶领域使用的视觉技术是根本不可靠的，给其它驾驶者和行人造成生命威胁，各个自动驾驶公司却吵着想让政府交通部门给他们大开绿灯。 某些公司被美国政府拒绝批准牌照之后大吵大闹， 骂政府监管部门不懂他们的 “高科技”， 太保守， 跟不上时代。有的公司更是异想天开， 想要政府批准他们的自动车上不安装方向盘，油门和刹车，号称自己的车已经不需要人类驾驶员， 甚至说 “只有完全去掉了人类的控制， 自动车才能安全运行。”



BE QUICK ABOUT IT

The letter is in response to a request for public comment by the NHTSA to a proposal it made last May to amend the Federal Motor Vehicle Safety Standards, a list of 75 rules that automakers must follow before selling cars to customers. Currently, those rules state that cars need to have controls such as a steering wheel and pedals.

But self-driving cars may not need these controls, proponents say, and the rules could be a hindrance to the technology being widely released at scale. Waymo and others like Cruise, the self-driving division of GM, and Ford hope to inevitably release tens of thousands of driverless cars without any human controls. Only by cutting the human completely out of the equation can an autonomous vehicle operate safely, these companies argue. And the NHTSA is considering rewriting the rules so self-driving car companies like Waymo can release cars without those features.

Waymo's letter is full of language like "promptly," "should move rapidly," and "urges NHTSA not to await" the completion of other third-party research into autonomous technology. The message it sends is one of urgency: the government needs to drop everything and change the damn rules already.

图 36.

一出出的闹剧上演，演得好像自动驾驶就快实现了，大家都在拼命抢夺这个市场似的，催促政府放宽政策。很是有些我们当年大炼钢铁，超英赶美的架势。这些公司就跟小孩子耍脾气要买玩具一样，全都吵着要爸妈让他玩自动驾驶，各种蛮横要求，马上给我，不然你就是不懂高科技，你就是“反智”，“反 AI”，你就是阻碍历史进步！给监管机构扣各种帽子，却完全不理解里面的难度，伦理和责任。玩死了人，却又抬出各种借口，不想负责任。

虽然 Tesla 和 Uber 是应该被谴责的，但这里面的视觉问题不只是这两家公司的问题，整个自动驾驶的领域都建立在虚浮的基础上。我们应该清楚地认识到，现有的所谓 AI 根本没有像人类一样的视觉理解能力，它们只是非常粗糙的图像识别，识别率还远远达不到人类的水平，所以根本就不可能实现自动驾驶。

什么 L1～L4 的自动驾驶分级，都是瞎扯。根本没法实现的东西，分了级又有什么用呢？只是拿给这些公司用来忽悠大家的口号，外加推脱责任的借口而已。出事故前拿来做宣传：“我们已经实现 L2 自动驾驶，目前在研究 L3 自动驾驶，成功之后我们向 L4 进军！”出事故后拿来推脱责任：“我们只是 L2 自动驾驶，所以这次事故是理所当然，不可避免的！”

如果没有视觉理解，依赖于图像识别技术的“自动驾驶车”，是不可能在复杂的情况下做出正确操作，保障人们安全的。机器人等一系列技术，也只能停留在固定场景，精确定位的“工业机器人”阶段，而不能在复杂的自然环境中行动。

要实现真正的语言理解和视觉理解是非常困难的，可以说是毫无头绪。一代又一代的神经学家，认知科学家，哲学家，为了弄明白人类“认知”和“理解”到底是怎么回事，已经付出了许多的努力。可是直到现在，对于人类认知和理解的认识都不足以让机器具有真正的理解能力。

真正的 AI 其实没有起步，很多跟 AI 沾点边的人都忙着忽悠和布道，没人关心其中的本质，又何谈实现呢？除非真正有人关心到问题所在，去研究本质的问题，否则实现真的理解能力就只是空中楼阁。我只是提醒大家不要盲目乐观，不要被忽悠了。与其夸大其词，欺骗大众，说人工智能快要实现了，不如拿已有的识别技术来做一些有用的事情，诚实地面对这些严重的局限性。

我并不是一味否定识别技术，我只是反对把 “识别” 夸大为 “理解”， 把它等同于 “智能”， 进行不实宣传，用于超出它能力的领域。诚实地使用识别技术还是有用的，而且蛮有趣。 我们可以用这些东西来做一些很有用的工具，辅助我们进行一些事情。 从语音识别，语音合成，图片搜索，内容推荐，商业金融数据分析，反洗钱，公安侦查，医学图像分析，疾病预测，网络攻击监测，各种娱乐性质的 app..... 它确实可以给我们带来挺多好处，实现我们以前做不到的一些事情。

另外虽然各公司都在对他们的 “AI 对话系统” 进行夸大和不实宣传，可是如果我们放弃 “真正的对话”， 坦诚地承认它们并不是真正的在对话，并没有智能，那它们确实可以给人带来一些便利。现有的所谓对话系统，比如 Siri, Alexa, 基本可以被看作是语音控制的命令行工具。 你说一句话，机器就挑出其中的关键字，执行一条命令。 这虽然不是有意义的对话，却可以提供一些方便。特别是在开车不方便看屏幕的时候，语音控制 “下一首歌”， “空调风量小一点”， “导航到最近的加油站” 之类的命令，还是有用的。

但不要忘记，识别技术不是真的智能，它没有理解能力，不能用在自动驾驶，自动客服，送外卖，保洁阿姨，厨师，发型师，运动员等需要真正 “视觉理解” 或者 “语言理解” 能力的领域，更不能期望它们取代教师，程序员，科学家等需要高级知识的工作。机器也没有感情和创造力，不能取代艺术家，作家，电影导演。所有跟你说机器也能有 “感情” 或者 “创造力” 的都是忽悠，就像现在的对话系统一样，只是让人以为它们有那些功能，而其实根本就没有。

你也许会发现，机器学习很适合用来做那些不直观，人看不透，或者看起来很累的领域，比如各种数据分析。实际上那些就是统计学一直以来想解决的问题。 可是视觉这种人类和高等动物的日常功能，机器的确非常难以超越。如果机器学习领域放弃对 “人类级别智能” 的盲目追求，停止拿 “超人类视觉” 一类的幌子来愚弄大众，各种夸大，那么他们应该能在很多方向做出积极的贡献。