

Lesson 2

[Video\(YouTube\)](#) / [Video\(bilibili\)](#) / [Lesson Forum](#) / [General Forum](#)

深入计算机视觉

深入学习计算机视觉应用，介绍一些令人惊奇的这周或者以后使用的东西。

论坛使用提示和技巧 [0:17]

两个重要的话题：

- [FAQ, 资源, 官方课程更新](#)
- [第二课官方资源和更新](#)

"Summarize This Topic"总结话题 [2:32]

才过了一周，最受欢迎的帖子已经有了1100个回复，这是一个可怕的数字。你不必阅读所有内容，你应该点击"Summarize This Topic"按钮，只让得到最多赞的回复显示出来。

The screenshot shows a forum post for "Lesson 1 chat".

- Statistics:** created 9d, last reply 10h, 1.1k replies, 5.8k views, 283 users, 1.5k likes, 50 links.
- Frequent Posters:** A grid of user icons with their names and reply counts: 63, 44, 37, 27, 25, 24, 20, 19, 16, 14, 14, 14, 12, 12, 11, 11, 11, 11, 10, 10, 9, 8, 8, 8.
- Popular Links:**
 - 104 Practical Deep Learning for Coders, v3 | fast.ai course v3 fast.ai
 - 91 Another data science student's blog – The 1cycle policy sgugger.github.io
 - 70 YouTube youtube.com
 - 55 Deep Learning ver3 Lesson 1 – Vikas Jha – Medium medium.com
 - 48 python - Google Colaboratory: misleading information about its GPU (only 5% RAM avail... stackoverflow.com
- Summary:** There are 1066 replies with an estimated read time of 87 minutes. A blue button labeled "Summarize This Topic" is visible.

继续运行 [3:19]

<https://course-v3.fast.ai/> 现在有一个“继续运行”的章节，它会告诉你（在每一个平台上）：

- 怎样保证你有最新的Notebook
- 怎样保证你有最新的fastai库

如果里面的方法对你不起作用，如果你遇到了一些麻烦的问题——我们都遇到过麻烦问题，你只需要把实例删除，再重新启动一个实例，除非你的实例上有非常重要的东西，这是摆脱麻烦的最简单的方法。

大家这周做了什么 [4:19]

分享你的作品

The screenshot shows a post on the fast.ai forum. The title is "Share your work here" with a checkmark icon. Below it, a message from "Part 1 v3" says "jeremy" and "Jeremy Howard (Admin)". The main message reads: "Show us what you've created with what you learned in fast.ai! 😊". A user named "astronomy88 Harold Nguyen" replies with a yellow profile icon, stating they were interested in voice recognition detection using Audacity to trim audio clips from Ben Affleck's speech in "The Boiler Room" and Joe Rogan and Elon Musk's podcast. They mention using 3 minutes and 30 seconds of audio from each. Another user, "chans.best chandan", replies with a green profile icon, mentioning they trained a model to clean up their WhatsApp downloaded images folder by manually classifying images from their own WhatsApp and Google search.

- [识别谁在说话——Ben Affleck还是Joe Rogan](#)
- [清除WhatsApp图片](#)

The screenshot shows a research paper titled "Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification" by Justin Salamon and Juan Pablo Bello, accepted in IEEE SIGNAL PROCESSING LETTERS in NOVEMBER 2016. The paper discusses the use of CNNs for environmental sound classification. It includes a figure showing classification accuracy for different models: SKM, PiczakCNN, SB-CNN, and the proposed model SB-CNN with augmentation. The proposed model shows significantly higher accuracy (around 0.80) compared to others (around 0.70). Below the paper, there is a spectrogram visualization of various sounds like gun_shot, children_playing, car_horn, drilling, street_music, engine_idling, siren, and jackhammer. A user named "etown Ethan Sutin" comments on the paper, stating they achieved 80.5% accuracy without augmentation, which is the state-of-the-art mean accuracy. They also mention that with extensive audio specific augmentation, their top accuracy was 79%, while without augmentation it was 74%.

论坛帖子

一个很有趣的项目是分析这个 [文章](#) 里的声音数据。在这个文章里，作者尝试识别声音类型。他们得到了接近80%的准确率，这时目前的最高成绩 (state of the art) 。Ethan Sutin尝试使用第一课里的方法来处理这个问题，得到了80.5%的准确率。这非常棒。据我所知，这是新的最好成绩。有可能有些人发表了更好的成绩，而我们还不知道，所以还不是很确定。但是我在twitter上讲了这件事，有很多人关

注我，如果有人知道更好的成绩，我认为他们会告诉我的。

[6:01]

State of the art on DHCD (देवनागरी)



Suvash Thapaliya @suvash · Oct 27

After lots of intentional tuneups, I can now announce that SoTA for accuracy on #DHCD #देवनागरी dataset is 99.02% (previously 98.50%). All thanks to @fastdotai library & courses, and the dataset creators @pk_gyawali amongst others.

#resnet34 #devanagari #deeplearning

Model fitting (transfer learning) on

_loss	valid_loss	accuracy
342	0.037567	0.989493
461	0.035617	0.990072
426	0.034761	0.990290

Training cycle (Accuracy = 99.02%)

character 26.yaw/character_4.gha / 9.93 / 0.98 character_23.ba/character_35.tra / 9.47 / 0.99 digit_3/character_36.gya / 9.32 / 0.93
character_25.mai/character_35.tra / 8.30 / 0.67 character_4.gha/character_17.tha / 7.87 / 1.00 character_21.pai/character_4.gha / 7.74 / 0.99
digit_0/digit_7 / 7.73 / 1.00 character_27.ra/character_11.taamatar / 7.70 / 1.00 character_20.hai/character_19.dha / 7.41 / 0.74

论坛帖子

Suvash 刷新了伽里字母识别的记录。我觉得他现在可以已经取得了比截图里更好的准确率。创建数据集的人在twitter上确认了这是最高成绩。他在twitter上说“噢，我创建了这个数据集。恭喜，你们创造了新的记录”。这真的很酷。

[6:28]

Alena Harley Follow · Oct 29 · 6 min read

The My Classif

Chapter

Approximat where they through the Determinin problems in dependent o

Cancer class because the mutations i

The diagram shows a central cell with various oncogenes and tumor-suppressor genes. Arrows point from external inhibitors to specific pathways: EGFR inhibitors target EGFR signaling; Cyclin-dependent kinase inhibitors target cyclin-dependent kinases; Immune activating anti-CTLA4 mAb targets the immune system; Telomerase Inhibitors target telomerase; Selective anti-inflammatory drugs target tumor-promoting inflammation; Inhibitors of VEGF signaling target angiogenesis; and Inhibitors of HGF/c-Met target invasion and metastasis. Other pathways shown include genome stability and mutation, replicative immortality, and tumor-promoting inflammation.

int mutation data using pathways, but how? What Gene2Vec encoding using information from gene

microarray in pathways.

起源之谜

我很喜欢Alena Harley的这篇文章。她讲述了很多癌细胞转移的细节和点变异的使用，以及为什么这是一个有挑战的重要问题。她有一些能她想用这个做什么的很好的图片，也展示了怎样可以把这些转换成图片。这是一个很酷的技巧，就像把声音转换成图片然后用第一课里的方法处理一样。这里把点变异转换成图片然后使用第一课里的方法。看起来她取得了新的记录，比之前的最高记录提升了30%。在Twitter上，一个生物分析公司的副总也看到了这个，也认为这是目前的最好成绩。这很振奋人心。

上周我们讲到这个简单的过程可以做很多事，确实如此。我要说像这样的实践使用了很多领域知识，比如说定义要生成的图片。我并不知道怎样做这个，因为我不知道点变异是什么，更不用说把它可视化，转换成CNN可以识别出意义的图片。但是深度学习的部分是没什么问题的。

[8:07]

Won at Science Hack Day

→ C https://simonwillison.net/2018/Oct/29/transfer-learning/

Simon Willison's Weblog

Automatically playing science communication games with transfer learning and fastai

This weekend was the 9th annual [Science Hack Day San Francisco](#), which was also the 100th Science Hack Day held worldwide.

Natalie and I decided to combine our interests and build something fun.

I'm currently enrolled in Jeremy Howard's [Deep Learning](#) course so I figured this was a great opportunity to try out some computer vision.

Natalie runs the [SciComm Games calendar](#) and accompanying [@SciCommGames](#) bot to promote and catalogue science communication hashtag games on Twitter.



另外一个很酷的作品来自Simon Willison和Natalie Downe，他们在这个周末创建了一个识别美洲豹的web应用程序，赢得了旧金山Science Hack Day。这很了不起。有很多人做了有趣的作品。这会激励大家创新思考。使用学过的知识来处理问题很酷。他们做的事情令人惊奇。需要说明的是，有成千上万的人在参加这个课程，我只挑选了一些比较令人惊奇的例子。事实上，Simon也是一个和我们上节课讲到的Christine Payne一样的令人妒忌的人，他们能把所有事都做的很出色，现在他在深度学习上也很出色。好吧，Simon可以在他学习深度学习的第一周赢得一个黑客马拉松，或许你们需要学习两周来赢得第一个黑客马拉松。

[9:22]



James Dellinger [Follow](#)

Oct 29 · 13 min read

If I Can You Can (and you should!)

My machine learning and deep learning journey almost ended here. It began back in early January of this year, when I saw the following page on scikit-learn.org. I momentarily and incorrectly assumed I wasn't "technical" enough to grasp machine learning because I couldn't immediately understand the mathematical notation that was pasted into articles all over the website.



Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable y and dependent feature vector x_1 through x_n :

$$P(y | x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n | y)}{P(x_1, \dots, x_n)}$$

Under the assumption that

$$x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n = P(x_i | y),$$

$$P(y | x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^n P(x_i | y)}{P(x_1, \dots, x_n)}$$

Given the input, we can use the following classification rule:

$$P(y | x_1, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i | y)$$

↓

$$\hat{y} = \arg \max_y P(y) \prod_{i=1}^n P(x_i | y),$$

Now I support math just as much as the next individual, but contrary to the impression given by photos on websites of certain AI startups, you don't actually have to derive gradient descent from scratch on a whiteboard before you can build a deep learning model that'll solve a problem you face.

Suffice to say, if I can do this, so can you. And you should. Because the world needs more real people who face real-world problems to begin experimenting with deep learning tools that might just help to solve those problems.

我觉得这件事也很值得讲，这周James Dellinger发了一篇讲他如何用第一课里的方法做鸟类分类的博文。引起我注意的是末尾内容，他说之前他觉得他没法做深度学习，因为他浏览了scikit-learn网站，这是一个最重要的python机器学习库，觉得深度学习不是他能做得到的，这些公式他理解不了。后来他意识到，不学习公式也能做出一些有用的东西。我认为这是一个很酷的信息。

[10:01]

Daniel.R.Armstrong Daniel Armstrong [Quote](#) 5d

What is the best way to contribute to the fast.ai library? When I was reading the fast.ai developer docs, I found the whole process a little overwhelming. I am wondering if anyone has any best practices other than using hub as Jeremy described above.

Daniel.R.Armstrong Daniel Armstrong [Quote](#) 4d

Unfortunately I can't tell you the specific parts of the developer docs that caused my confusion. I think my problem is I don't have any real world experience with all the nuances of git, and version controls. Frankly I had no idea how much there was so much to it, I think it just caught me off guard. I thought it was just push, pull, and clone. Reading the dev docs is kind of like reading a different language, my brain just shut down. 🐾. I will learn it all soon, but I will have to put in some time to learn it. Maybe then I can give you a better answer.

One of the reasons I love the fast.ai way, is it forces me to learn so much more than just the fast.ai library. I have learned about bash, terminal, linux, notebooks, curl, AWS, computer hardware, vim, tmux, debugger, pytorch, and so much more.

Daniel.R.Armstrong Daniel Armstrong [Reply](#) jeremy 3d

I am happy to say that I submitted my first PR this morning, and it was accepted/merged this morning! I was so excited!

我想在论坛上高亮Daniel Armstrong。我觉得他是一个榜样。他说，我想向库提交贡献，我看了文档，它很大。第二天的第二条消息里，他说，所有东西我都不懂，我不知道这要花多久，我应付不了，我的大脑要停机了，但我喜欢这驱使我学习这么多东西。第三天，他说，我刚刚提交了我的第一个pull request。我觉得这很棒，感觉到害怕并没有关系，确实有很多东西要学。你只要选择其中一部分，慢慢深入进去就好。尝试提交对一段代码或者一段文档的更新，或者创建一个分类器或者其他的东西。

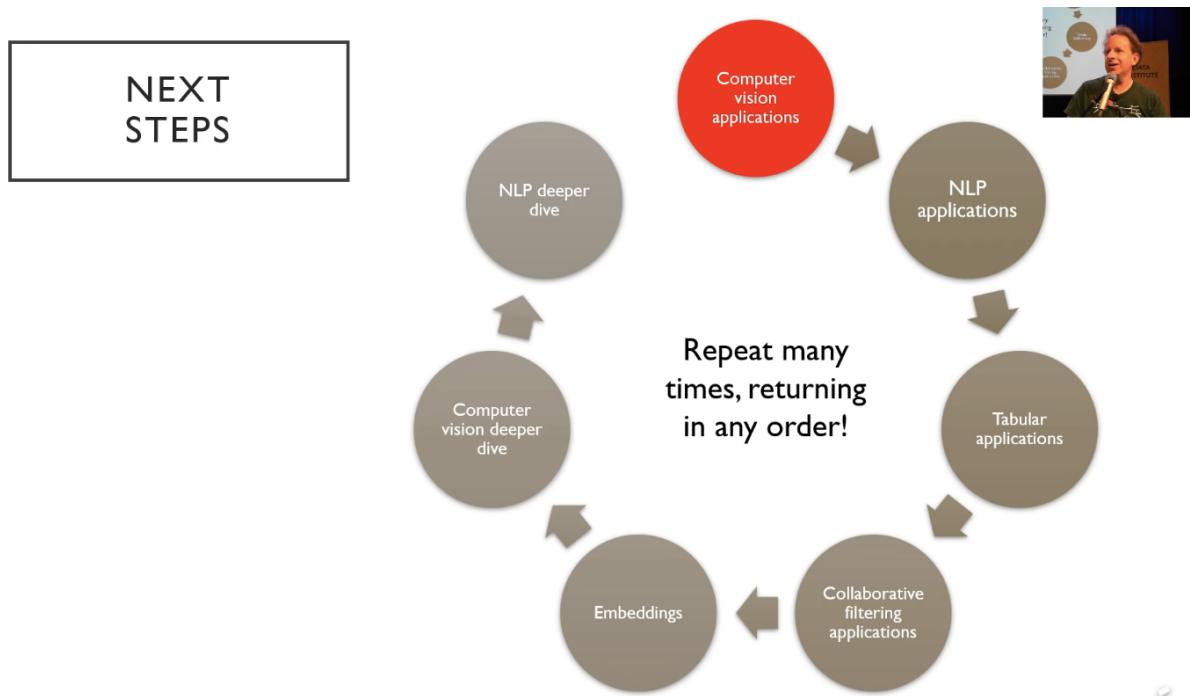
[10:49]

大家创建了很多很酷的分类器。这很振奋人心。

- 特立尼达和多巴哥岛民 VS 假面舞者 分类器
- 西葫芦 VS 黄瓜 分类器
- 上周的猫狗品种分类器。对它做一些探索工作看看最主要的特征是什么，发现其中一个是多毛的狗和没毛的猫。所以你可以去尝试解释很多有趣的问题。
- 论坛上另外一些人对动漫做了相同的事情，他们误打误撞做了一个动漫角色头发颜色分类器。
- 我们现在可以区分新的和旧的巴拿马巴士。
- Henri Palacci发现他可以以85%的准确率识别110个国家的卫星图，这明显超过了人类的表现。
- Batik的布料分类器的准确率是100%。
- Dave Luo做了这样有趣的事。他超前了一些，使用了我们会在接下来的课程里会学习的技术创建了一个程序，它可以识别卫星图里的建筑是完工的、未完工的还是仅有地基。

有很多很多有吸引人的项目。不必着急，这才过了一周。并不是说所有人都要做出一个项目。很多做出项目的学员参加过之前的课程，所以他们领先了一些。今天我们来学习怎样创建你自己的分类器。

[12:56]



今天开始，在我们深入学习怎样让这些计算机视觉分类器有效运行后，我们会学习用这些处理文本，然后处理表格数据，比如说excel电子表格和数据库。然后我们会学习协同过滤(比如推荐系统)。这会让我们学习到“embeddings（嵌入）”，这是这些应用下层的重要基础。然后我们回到计算机视觉，更深入地学习计算机视觉，然后更深入地学习自然语言处理。这样做，更有利子掌握这些内容。你会多次看到一个主题，而不是只学一次，就不会再在剩下的课程里看到它，几周后我们会回过头来再学习自然语言处理和计算机视觉这两个重要的应用。这会促使你记住这些知识，不会忘记它们。

[14:06]

IF YOU'RE STUCK, KEEP GOING!

Code first	Focus on learning from experiments
The whole game	It's like learning soccer as a kid (Perkins)
Concepts, not details	We'll gradually dig in to all the details
Do lesson 2	...even if you don't understand all of lesson 1

最开始很多人，尤其是有更多坚实科学背景的人，觉得这种“嘿，这是代码，输入它，运行它”的教学方式比讲很多理论的教学方式令人困惑，让人觉得惊奇、怪异。对有这种想法的人，我想建议你们进行下去。你不必记住所有内容，不必理解所有内容，不必理解这些是怎样运行的。希望你能输入代码，运行它，产生结果，然后你可以做些实验，这样你可以了解到它是怎样运行的。然后继续向前。很多完成了这个课程并取得了成功的人都看了这个视频至少3遍。基本上先快速过一遍，然后慢慢过一遍，然后再更慢地过一遍。我总会听到他们说“每次我看一遍，我都很学到很多新内容”。所以就算没有明白里面的内容，也不要在第一课停下来。

这种教学方式是基于很多学习理论方面的学术研究的。哈佛大学的David Perkins是一个学习理论的学者，他有一个了不起的比喻。如果你教一个孩子踢足球，你不会先教他球和草地之间的摩擦力，然后口头教他们传球，然后教他们当把东西踢到空中时的抛物线的数学原理。你只会说，这是一个球，我们来看别人踢球吧。好，我们来踢球。然后再在接下来的几年里逐步学习越来越多的技能，变得越来越擅长踢球。我们要教你做的和踢球是一样的，我们是学习敲代码，观察输入输出。

使用谷歌图片搜索做泰迪熊识别 [16:21]

让我们进入第一个notebook [lesson2-download.ipynb](#)。我们要学习怎样使用自己的图片数据创建自己的分类器。这很像我们上周的宠物分类器，但它可以识别你想要的任何东西。就像我们刚刚看到的例子那样，怎样从头创建你自己的巴拿马大巴识别程序。Adrian Rosebrock提供了帮助，他有一个很好的叫做[pyimagesearch](#)的网站，他做了一个详细的说明[怎样使用谷歌图片搜索创建深度学习数据集](#)。这对我们使用的一些技术提供了启发，感谢Adrian，你应该看看他的网站，里面有很多好资源。

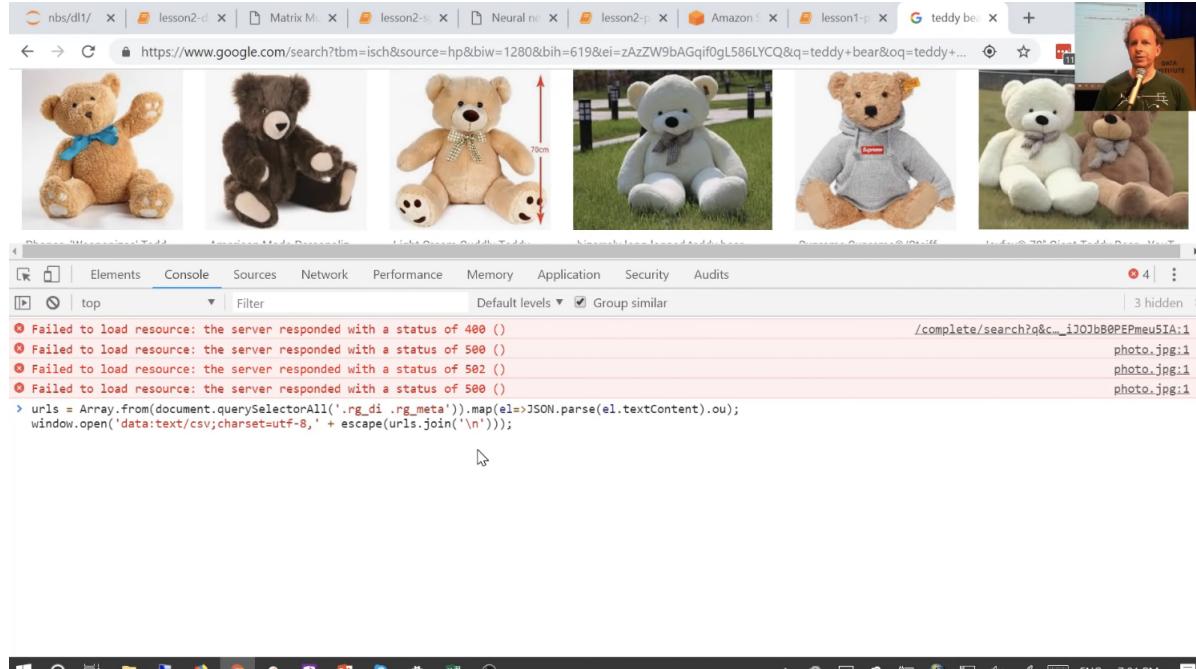
我们要创建一个泰迪熊识别程序。用它区分泰迪熊和黑熊、灰熊。这很重要，我有一个三岁的女儿，她需要知道她遇到的是什么。在我们家，你会对怪兽、狮子和其他怪物的数量感到吃惊，尤其是邻近万圣节的时候。我们需要经常看看，保证我们抱着的是一个真正的泰迪熊。让我们尽最大可能识别出来。

第一步：收集各类图片的url

我们要先找到一些泰迪熊的图片，我们先看看它们长什么样。我打开<https://images.google.com/>，输入Teddy bear，划动滚动条，直到找到一批合适的图片。看起来有很多泰迪熊。

然后回到[notebook](#)，你可以看到上面说“到谷歌图片，搜索，向下滚动”。下步要做的是，获取所有这些url的列表。要想做到这个，回到谷歌，在Windows/Linux下点击 [Ctrl](#) [Shift](#) [J](#)，在Mac下点击 [Cmd](#) [Opt](#) [J](#)，在显示出的窗口里贴入下面的代码：

```
1 urls = Array.from(document.querySelectorAll('.rg_di .rg_meta')).map(el=>JSON.parse(el.textContent).ou);
2 window.open('data:text/csv;charset=utf-8,' + escape(urls.join('\n')));
```



有些人可能没有用过Javascript，这是一个Javascript控制台。我点击回车，它下载了一个文件。我把它命名为teddies.txt，点击“保存”。好了，现在我有一个包含泰迪熊图片url的文件。然后对黑熊和灰熊做同样的操作，把它们放入对应名字的文件里。

第二步：下载图片 [19:39]

第二步是下载图片到服务器。要记得，当我们使用Jupyter Notebook时，它不是运行在我们的电脑上，它运行在SageMaker或者Crestle、或者Google cloud，等等。我们运行一些Jupyter单元格。来引用fastai库：

```
1 from fastai import *
2 from fastai.vision import *
```

先从黑熊开始。点击黑熊的单元格，运行它。这里有三个单元格在做关于不同信息的相同事情。这是我用Jupyter notebook时喜欢的一种方式。很多有科学背景的人对这种方式感到吃惊。这不是在做可复现的研究。我点击黑熊的单元格，运行它来创建一个叫做black的文件夹和一个叫做urls_black.txt的文件。我跳过后面两个单元格。

```
1 folder = 'black'
2 file = 'urls_black.txt'
```

```
1 folder = 'teddys'
2 file = 'urls_teddys.txt'
```

```
1 | folder = 'grizzly'  
2 | file = 'urls_grizzly.txt'
```

然后运行这个单元格来创建文件夹。

```
1 | path = Path('data/bears')  
2 | dest = path/folder  
3 | dest.mkdir(parents=True, exist_ok=True)
```

然后拉到下一节，运行下一个单元格，它会下载黑熊的图片。这只会下载黑熊图片到对应文件夹。

```
1 | classes = ['teddys', 'grizzly', 'black']  
  
1 | download_images(path/file, dest, max_pics=200)
```

现在我回来点击 `'teddys'`，滑下来重复相同的事情。这样，我来回滑动，下载每个想要的类别。手动的操作比较多，但是对我来说，这样重复实验几次，没什么问题。如果你擅长提前计划，你可以写一个合适的循环，或者写可以做类似事情的代码。当你在我的notebook里看到这种类似配置的单元格，这是一个明显的标志，这些不按顺序执行。我点击一个地方，再去另一个地方。对我来说，我是实验主义者。我很喜欢在notebook里做实验，我把它当成实验日志，我尝试执行，看看会发生什么。这就是我的notebook最后的样子。

这是一个有争议的话题。很多人觉得这不对，应该从上到下依次运行。所有你做的都应该可以重现。我觉得这不是发挥创造力的最好方式。我觉得做尝试，看看发生了什么，不停探索是激发创造力的最好方式。你可以看到什么是有效的。

这会下载图片到你的服务器。它使用了多线程。这样做有一个坏处，就是如果出了错，不太容易找到出错的位置。下面有一段注释掉的代码，设置了 `max_workers=0`。这样就不会启用多线程，会更容易定位问题。如果不能正常下载，试下使用这段代码。

```
1 | # If you have problems download, try with `max_workers=0` to see exceptions:  
2 | # download_images(path/file, dest, max_pics=20, max_workers=0)
```

第三步：创建ImageDataBunch [22:50]

下一步需要做的是删除那些不是图片的文件。这经常发生。每次都会有一些图片因为各种各样的原因出错。谷歌告诉我们这个url有一个图片，但无法下载到。我们在库里加入了一个 `verify_images` 方法，它会检查路径下的所有图片是否有问题。如果你传入 `delete=True`，它会删除有问题的图片。这是一个创建干净数据集的很好用的方法。

```
1 | for c in classes:  
2 |     print(c)  
3 |     verify_images(path/c, delete=True, max_workers=8)
```

teddys

100.00% [272/272 00:06]

grizzly

100.00% [168/168 00:05]

```
cannot identify image file '/data0/datasets/part1v3/bears/grizzly/00000011.jpg'  
cannot identify image file '/data0/datasets/part1v3/bears/grizzly/00000014.jpg'  
black
```

100.00% [176/176 00:05]

现在我有了一个熊的文件夹，里面包含了灰熊文件夹、泰迪文件夹、黑熊文件夹。我们有了创建 ImageDataBunch 的基本内容。让我们创建它来做深度学习吧。

现在，你下载到的大部分 kaggle 数据集或者学术数据集里都会包含训练文件夹、验证文件夹、测试文件夹，里面存放了不同的数据。在这个例子里，我们没有独立的验证集，因为我们仅仅是从谷歌下载了这些图片。但我们仍然需要一个验证集，否则你没办法知道你的模型的效果如何。我们花些时间来讲解下怎样做。

当创建 data bunch 时，如果你没有单独的训练集和验证集，你在参数里可以写训练集是在当前目录（因为默认的，它会去找叫做“train”的目录），我建议把 20% 的数据作为验证集。这会自动随机使用 20% 的数据作为验证集。你可以看到当我随机创建验证集时，我会在前面把随机种子设为固定值。这样，每次我运行这段代码时，我会得到相同的验证集。通常，我不会坚持让机器学习实验可复现（就是保证每次都得到完全相同的结果）。对我来说，随机性对验证你的模型是否稳定，是否每次都有效是很重要的。但是，保持验证集不变很重要。否则当你尝试调整超参数来改进模型时，如果你用了一个不同的验证集，你就没办法知道取得不同的结果是因为超参数改变引起的还是因为验证集变简单引起的。这就是每次都要设置随机种子的原因。

```
1 | np.random.seed(42)  
2 | data = ImageDataBunch.from_folder(path, train='.', valid_pct=0.2,  
3 |         ds_tfms=get_transforms(), size=224,  
4 |         num_workers=4).normalize(imagenet_stats)
```

[25:37]

现在我们得到了一个 data bunch，你可以查看下 `data.classes`，你可以看到这和我们创建的文件夹一致。它知道了要识别的类别（这些类别代表所有可能的标签）是黑熊、灰熊、泰迪熊。

```
1 | data.classes
```

```
1 | ['black', 'grizzly', 'teddys']
```

我们可以运行 `show_batch` 来看下。一些图片比较难区分。有些并不是照片。有的比较有趣，如果你遇到一个黑熊站在灰熊上，这就比较难分类。

```
1 | data.show_batch(rows=3, figsize=(7,8))
```



你可以在这里再检查一遍。`data.c` 属性表示数据集里有多少种可能的标签。晚些时候我们会学到 `c` 的其他含义。我们可以看到在训练集里有多少图片，验证集里有多少图片。现在训练集里有473张，验证集里有140张。

```
1 | data.classes, data.c, len(data.train_ds), len(data.valid_ds)
```

```
1 |(['black', 'grizzly', 'teddys'], 3, 473, 140)
```

第四步：训练模型 [26:49]

现在，我们要用这个数据来创建卷积神经网络。我使用resnet34。设置每次迭代打印出错误率。

```
1 | learn = create_cnn(data, models.resnet34, metrics=error_rate)
```

然后运行 `fit_one_cycle` 4次，看下结果。错误率是2%。这很不错。有时区分黑熊和灰熊比较简单，有时就有些难。这个分类器看起来区分得不错。

```
1 | learn.fit_one_cycle(4)
```

```
1 | Total time: 00:54
2 | epoch  train_loss  valid_loss  error_rate
3 | 1      0.710584    0.087024    0.021277    (00:14)
4 | 2      0.414239    0.045413    0.014184    (00:13)
5 | 3      0.306174    0.035602    0.014184    (00:13)
6 | 4      0.239355    0.035230    0.021277    (00:13)
```

我喜欢把开始的结果保存下来，这样下次再使用时就不用再花54秒来运行它。

```
1 | learn.save('stage-1')
```

像通常一样，我们解冻模型的其他部分。在这个课程里，我们会学习更多关于解冻的知识。

```
1 | learn.unfreeze()
```

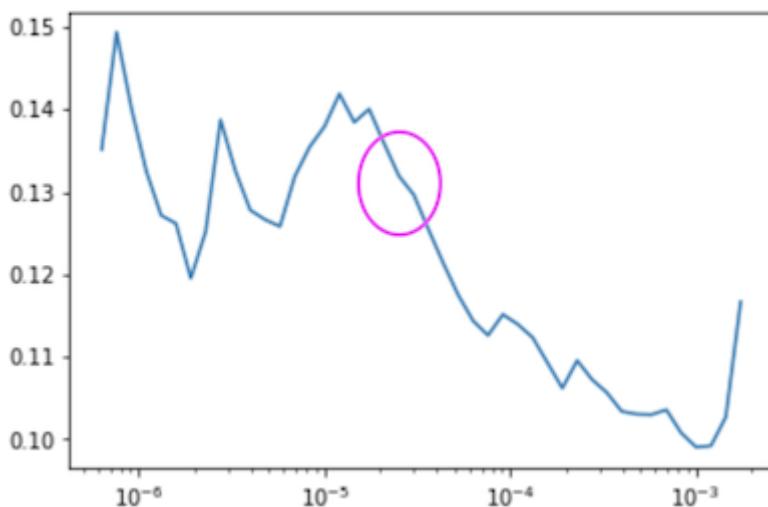
然后我们运行学习率查找方法，把它的结果画出来（它会告诉你输入什么代码来画出结果）。我们看下这个图。

```
1 | learn.lr_find()
```

```
1 | LR Finder complete, type {learner_name}.recorder.plot() to see the graph.
```

今天我们要学习下学习率。先讲下这个。在learning rate finder的图形里，需要找的是这个最长的向下的斜坡。想找到有效的部分，需要多练习来找到些感觉。如果你不确定，就多试几个，看看哪个运行得更好。我试了一下，觉得这部分 (10^{-5} 到 10^{-3} 之间) 最好。我会选择这里的作为学习率。[28:28].

```
1 | learn.recorder.plot()
```



这里能看到，我选择 $3e-5$ 作为最小的学习率。我会选择 $1e-4$ 或者 $3e-4$ 作为最大学习率。一般情况下，这会运行的很好。你需要知道，大部分参数的影响没有那么大。如果你只是照抄我每次使用的数值，它也会很有效。我们今天也会讲到在什么情况下它会不起作用。

```
1 | learn.fit_one_cycle(2, max_lr=slice(3e-5,3e-4))
```

```
1 | Total time: 00:28
2 | epoch  train_loss  valid_loss  error_rate
3 | 1      0.107059   0.056375   0.028369   (00:14)
4 | 2      0.070725   0.041957   0.014184   (00:13)
```

在运行完两个epoch后，我们得到了1.4%的错误率，这看起来很不错。我们在谷歌上下载了一些图片，创建了一个分类器，得到了1.4%的错误率。我们把它保存下来。

```
1 | learn.save('stage-2')
```

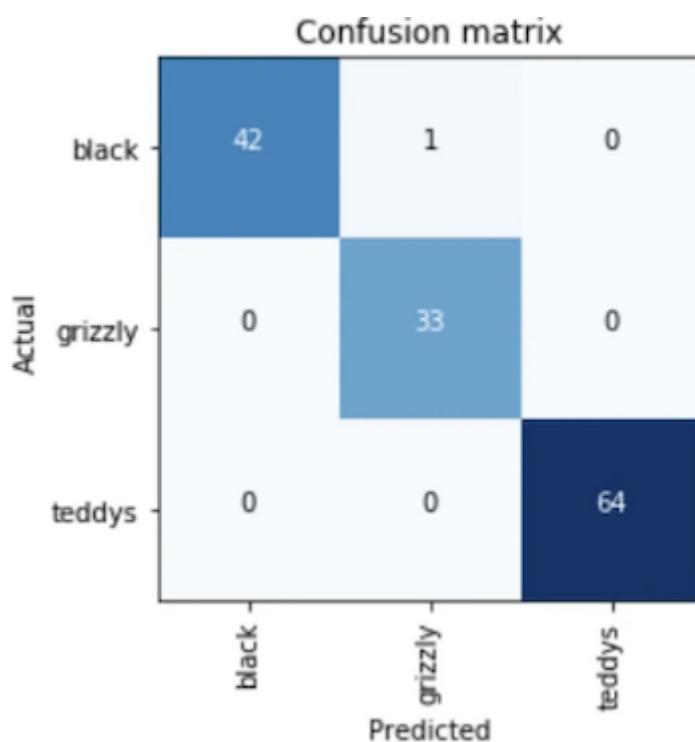
解释 [29:38]

像以前一样，我们可以使用ClassificationInterpretation类型来看看结果。

```
1 | learn.load('stage-2')
```

```
1 | interp = ClassificationInterpretation.from_learner(learn)
```

```
1 | interp.plot_confusion_matrix()
```



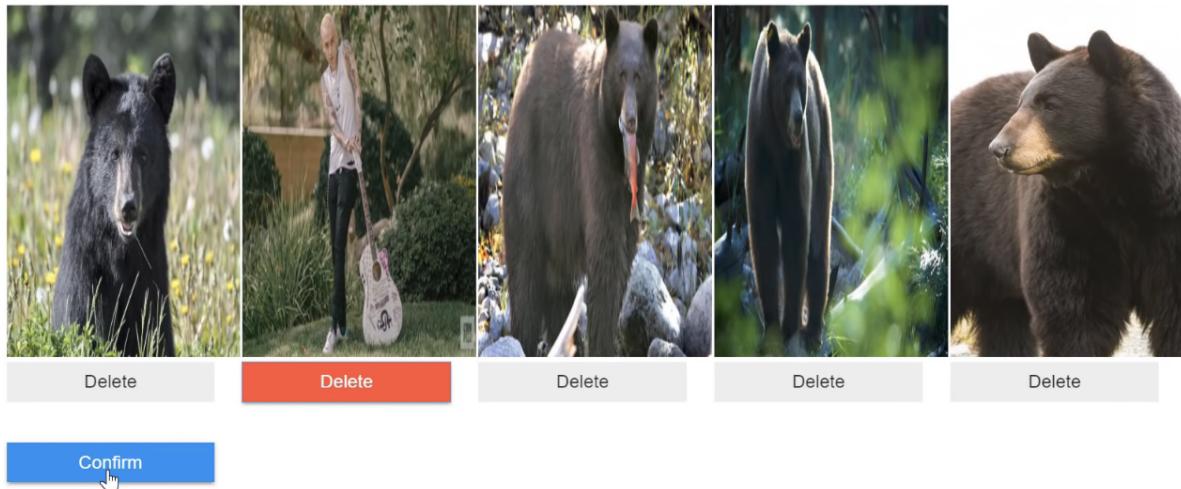
这里有一个错误。有一个黑熊被当成了灰熊。这是一个有用的步骤。我们取得了很大进步。如果我们的数据集更准确的话，我们可能能做得更好。谷歌图片搜索可能不能每次都给出正确的图片。我们怎样解决它呢。我们把它清除掉。用深度学习程序帮助人类专家是一个好主意。非常少的人讲这个，但是对我来说，这是最有用的技巧，尤其是对你们来说。大部分学习这个课程的人是领域专家，不是计算机科学专家。这里你可以使用你关于生物学里点变异或者巴拿马大巴或者其他方面的知识。来看下怎样做。记得上节课哪个画出最大损失的图吗，我们可以看到错得最多的，最不确定的数据。如果数据被标错了，不太可能会被识别正确。这不太可能发生。所以我们要关注那些模型不确定的或者确定但搞错的数据，这些数据可能是被标错了。

这周旧金山fastai学习小组创建了这个叫做FileDelete的组件。Zach、Jason、Francisco做了这个工具，我们可以取到解释对象（interp）里的损失度最高的数据。不仅有`plot_top_losses`方法，也有`top_losses`方法。`top_losses`返回最差的损失度，和最差数据的索引。如果你没有输入任何参数，它会返回排过序的整个数据集，第一个有最高的损失度。fastai的每一个数据集都有`x`和`y`，`x`是用来获取图片的数据，就是文件名，`y`是标签。如果我们得到了索引，把它传入数据集的`x`，这会返回按照损失度（确信但分错类或者不确定）排序的文件名。我们可以把这个传入这新的组件里。

说明下，这个`top_loss_paths`包含数据集里所有的文件名。这里说的“数据集”是指验证集。我们要清理掉标记错的图片，或者不应该在这里的图片，我们要把这些从验证集里删除，这样我们的成绩才会更准确。然后你需要再做一次，把`valid_ds`换成`train_ds`来清理训练集，把干扰数据去掉。这是很好的做法。稍后我们也会讲到测试集，如果你有测试集，你也应该在上面做一遍。

```
1 | from fastai.widgets import *
2 |
3 | losses, idxs = interp.top_losses()
4 | top_loss_paths = data.valid_ds.x[idxs]
```

```
1 | fd = FileDelete(file_paths=top_loss_paths)
```



我们运行`FileDelete`，传入排序过的路径，出现的内容是和`plot_top_losses`一样的。也是那些错的或者不确定的数据。没什么疑问，这个图片（左数第二张）不是泰迪熊或者黑熊灰熊。它不应该出现在数据集中。我点击删除按钮。剩下的看起来都是熊，我点击确认按钮，程序会显示出另外5张图片。

我会一直检查这些图片，直到连续几屏，里面的图片都是正常的时候才会停下来。现在可以对训练集做同样的操作，然后重新训练模型。

我们的旧金山学习小组做的是一个运行在Jupyter notebook里的小程序，可能你们没有注意到这点。这不仅是可以实现的，也超乎意料得简单。你可以输入两个问号来查看详情。这是它的代码。

```

In [17]: ??FileDeleter
In [16]: fd = FileDeleter(file_paths=top_loss_paths)

Init signature: FileDeleter(file_paths:Collection[Union[pathlib.Path, str]], batch_size:int=5)
Source:
class FileDeleter():
    "Flag images in `file_paths` for deletion and confirm to delete images, showing `batch_size` at a time."
    def __init__(self, file_paths:Collection[PathOrStr], batch_size:int=5):
        self.all_images,self.batch = [],[]
        self.batch_size = batch_size
        for fp in [o for o in map(Path,file_paths) if o.is_file()]:
            img = self.make_img(fp)
            delete_btn = self.make_button('Delete', file_path=fp, handler=self.on_delete)
            self.all_images.append((img, delete_btn, fp))
    self.render()

```

如果你之前做过GUI编程，这看起来很普通。里面有几个你点击按钮时会触发的回调，里面执行了一些标准的python代码，为了渲染它你需要使用widget，使用标准的box来放置它。这个在notebook里创建应用的主意可能大才小用了，但它非常不错，它能让你为你的同伴创建工具。事实上，当你在看这个视频时，你可能会发现notebook里有很多个按钮，因为我们现在已经整理了很长的一份待办任务清单来添加这些。

我希望你们知道，在notebook里写一个应用程序是可能的，如果你搜索“[ipywidgets](#)”，你可以了解这个轻量级的GUI框架，知道你可以创建什么样的小部件，它们长什么样，是如何工作的，等等。这些都可以用在你的模型上。这不是一个为生产环境提供程序的好方法，因为它是基于notebook的。这主要是用来帮助大家来做实验的。对于生产环境的程序，你需要真正地构建一个web应用，下面我们来看看如何做。

把模型放在生产环境 [37:36]

在你清理掉有问题的图片后，你可以重新训练模型，希望模型准确率能变得更高些。大部分时候，可能并没有作用，这可能会让人觉得奇怪。总的来说，这些模型能够接受一定数量干扰数据。如果你的干扰数据不是随机的，而是在某些分类里比较多，而另一些比较少，那可能会带来问题。如果你完成了清理数据的过程，再次训练模型，会发现它比以前变好了0.001%，这很正常。不过保证你的数据集里没有太多有偏向的干扰数据总是一个好的做法。

现在，我们可以着手把模型放到生产环境了。很多人问我应该用谷歌还是Facebook的分布式服务，怎样同时在上千个GPU上运行程序。对于绝大部分人来说，你应该用CPU，而不是GPU。为什么？因为CPU更擅长同时做很多事情。除非你有一个访问量非常大的网站，你不太可能同时有64张图片需要分类，把它们一起放入GPU。。如果你有这样的访问量，用GPU处理它们时，你需要做队列来组成一个批次后同时处理。。所有用户需要等待，直到凑满了一个批次再执行，这很麻烦。如果你还想扩容，又是一个大麻烦。如果你把它封装起来在CPU上执行会简单很多。这确实比GPU会花费10多倍的时间，大概是从0.01秒变成0.2秒。我们讨论的时间基本在这个数量级。但它很容易扩容，你可以把它放在任何的标准托管服务基础设施上。这很便宜，可以方便地水平扩容。所以大部分我认识的人都在CPU上运行他们的基于深度学习的应用。我们要用的名词是“推理（inference）”。当你不是在训练一个模型，而是用训练好的模型来做预测时，我们把这叫做推理。所以我们这样讲：

你应该用CPU来做推理

在做推理时，你有了预训练的模型，你保存了那些权重，你怎样才能用它们做出像Simon Willison的美洲狮检测程序一样的东西？

首先你需要知道你训练的类别是什么。还需要知道他们的顺序。所以你需要为它们排序或者直接输入它们，或者用其他的方式，总之要保证和你训练时顺序的完全一致。

```
1 | data.classes
```

```
1 | ['black', 'grizzly', 'teddys']
```

如果你的服务器上没有GPU，它会自动使用CPU。如果有GPU，但现在想用CPU来测试，你可以之间取消掉这行的注释，这会把这个参数传给pytorch，让fastai来使用CPU。

```
1 | # fastai.defaults.device = torch.device('cpu')
```

[41:14]

这是一个例子。我们的不是美洲狮检测程序，是一个泰迪熊检测程序。我的女儿Claire用它来决定要不要拥抱图片里的东西。她用爸爸的深度学习模型，她找到一张图片，这是她上传到web程序里的图片，这是一个潜在的拥抱对象的图片。我们把这存在一个叫的对象里，在fastai里使用open_image来打开一个图片，很有趣。

```
1 | img = open_image(path/'black'/'00000021.jpg')
2 | img
```



这是我们之前保存的类别的列表。通常我们创建一个data bunch，这次我们不会再从放了很多图片的文件来创建data bunch，我们要创建一种特殊的data bunch，它一次只接收一张图片。所以现在我们不传入任何数据。我们传入路径的唯一原因是让它知道在哪里加载模型。这是存放模型的路径。

我们需要做的是，传递数据要和训练数据有相同的格式。相同的变形（transforms）、相同的尺寸（size）、相同的正则化（normalization）。我们以后会深入学习这些。在使用前确保它们的格式是相同的。

现在你有了一个没有任何数据的data bunch。它可以用和训练时一样的方式来转换一张新图片。现在就可以来做预测了。

现在可以使用这个空的data bunch来create_cnn，你需要使用你训练完的模型。现在可以加载之前保存的权重，这只需要在web app启动时做一次。运行这行代码要0.1秒。

```
1 | classes = ['black', 'grizzly', 'teddys']
2 | data2 = ImageDataBunch.single_from_classes(path, classes,
3 |     tfms=get_transforms(), size=224).normalize(imagenet_stats)
4 | learn = create_cnn(data2, models.resnet34)
5 | learn.load('stage-2')
```

然后就可以 `learn.predict(img)`，很好，我们成功了。它不是一个泰迪熊，确实是一个黑熊。多亏了这个准确的深度学习模型，我的女儿不会和黑熊做尴尬的拥抱了。

```
1 | pred_class,pred_idx,outputs = learn.predict(img)
2 | pred_class
```

```
1 | 'black'
```

怎样用于生产环境呢？我拿了[Simon Willison的代码](#)，剽窃可耻，这可能不太好，不过生产环境的代码基本就是这样的。Simon使用了一个很酷的web app工具箱[Starlette](#)。如果你使用过Flask，这看起来很相似，不过它更现代化一些，现代化是指你可以使用 `await`，这代表你在等待一些耗时的工作(比如抓取一些数据)时不会阻塞进程。做类似预测或者加载数据之类的工作时，能使用Python 3里现代的异步方式是很棒的。强烈推荐使用Starlette来创建web app。

你像往常一样创建一个路由，你声明它是 `async`，来让它在等待时不会阻塞进程。

你打开图片，调用 `learner.predict`，返回响应。然后你可以使用Javascript客户端或其他东西来显示它。好了，这基本是你的web程序里的主要内容。

```
1 | @app.route("/classify-url", methods=["GET"])
2 | async def classify_url(request):
3 |     bytes = await get_bytes(request.query_params["url"])
4 |     img = open_image(BytesIO(bytes))
5 |     _,_,losses = learner.predict(img)
6 |     return JSONResponse({
7 |         "predictions": sorted(
8 |             zip(cat_learner.data.classes, map(float, losses)),
9 |             key=lambda p: p[1],
10 |             reverse=True
11 |         )
12 |     })
```

这周练习下这个。如果你之前没有创建过web程序，网上有很多不错的教程和初始代码。如果拿不准，不妨试下Starlette。有些免费的托管服务可以用，比如[PythonAnywhere](#)。Simon用的是[Zeit Now](#)，你可以像docker一样打包然后上传，在服务器上启动。它不会有什麼费用，你可以把你创建的分类器变成web应用了。很乐于看到你们能够做到这些，这很有趣。

https://course-v3.fast.ai/deployment_zeit.html

会出错的地方 [46:06]

我讲过大多数时候，我展示的这些经验法则是有效的。如果你看看“分享你的作品”板块，可以看到很多时候，人们说“我下载了这些图片，试着做了这些，运行得比我预期的好”，这很酷。另外有1/20的人说我遇到了问题。我们来看下你遇到的问题是怎么回事。这里我们开始介绍一些理论，来理解为什么我们会遇到这些问题和如何修复这些问题，它能帮助我们知道一些运行的机制。

首先，看下可能的问题，基本上是：

- 你的学习率太高或者太低

- 你的迭代次数太多或者太少

我们学习下这是什么意思，为什么会有影响。首先，作为实验主义者，先试着模拟下

学习率太高

修改下泰迪检测程序，把学习率设到很高。默认值是0.003，大部分时间是有效的。如果使用0.5的学习率会怎样。这很大。会发生什么。我们的验证损失度变得非常高。记住，这个值是要小于1的。如果你看到你的验证损失度这么高，即使我们还没有学什么是验证损失度，你也要知道，如果出现了这种情况，是你的学习率太高了。知道这些就够了。把它改小些。不管你的epoch次数是多少。如果发生了这种情况，你需要重新创建神经网络，使用一个更低的学习率来训练它。

```
1 | learn = create_cnn(data, models.resnet34, metrics=error_rate)
```

```
1 | learn.fit_one_cycle(1, max_lr=0.5)
```

```
1 | Total time: 00:13
2 | epoch  train_loss  valid_loss  error_rate
3 | 1      12.220007  1144188288.000000  0.765957    (00:13)
```

学习率太低 [48:02]

如果把学习率从0.003改成 $1e-5$ (0.00001)会怎样?

```
1 | learn = create_cnn(data, models.resnet34, metrics=error_rate)
```

这是使用默认学习率的结果：

```
1 | Total time: 00:57
2 | epoch  train_loss  valid_loss  error_rate
3 | 1      1.030236  0.179226  0.028369    (00:14)
4 | 2      0.561508  0.055464  0.014184    (00:13)
5 | 3      0.396103  0.053801  0.014184    (00:13)
6 | 4      0.316883  0.050197  0.021277    (00:15)
```

一个epoch，能得到2%到3%的错误率。

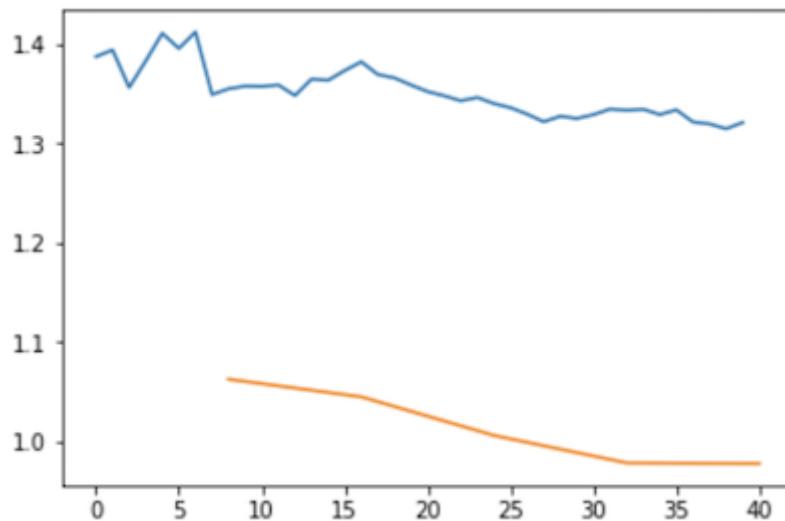
使用很低的学习率，错误率逐渐改善，但很改善地很慢。

```
1 | learn.fit_one_cycle(5, max_lr=1e-5)
```

```
1 | Total time: 01:07
2 | epoch  train_loss  valid_loss  error_rate
3 | 1      1.349151  1.062807  0.609929    (00:13)
4 | 2      1.373262  1.045115  0.546099    (00:13)
5 | 3      1.346169  1.006288  0.468085    (00:13)
6 | 4      1.334486  0.978713  0.453901    (00:13)
7 | 5      1.320978  0.978108  0.446809    (00:13)
```

你可以画出它来。`learn.recorder` 是一个保存训练过程日志的对象。你可以调用 `plot_losses` 来画出验证集和训练集的损失度。你可以看到它们下降的很慢。如果出现了这种情况，就代表你的学习率太低了。把它提高10倍或者100倍，再训练一遍。如果学习率太低，另一个现象是训练集的损失度比验证集损失度高。这意味着还没有完全拟合，通常是因为学习率太低或者epoch次数太少。所以你的模型出现了这样的情况，要多训练几个epoch或者提高学习率。

```
1 | learn.recorder.plot_losses()
```



如果花了很长时间，它看了每个图片太多次，有可能过拟合。

epoch次数太少 [49:42]

如果我们只训练一个epoch会怎样？我们的错误率比随机值要好，只有5%。但是看下这里，看下训练集损失度和验证集损失度的区别——训练集损失度比验证集高很多。epoch次数太少和太低的学习率的现象看起来很像。你可以试着多运行几个epoch，如果还是没有效果，就用一个更高的学习率。如果学习率太高，损失度超过了十万，就把学习率调低，少运行几个epoch，找到平衡。在大部分情况下，这样就够了。只有二十分之一的概率，默认值不起作用。

```
1 | learn = create_cnn(data, models.resnet34, metrics=error_rate,  
pretrained=False)
```

```
1 | learn.fit_one_cycle(1)
```

```
1 | Total time: 00:14  
2 | epoch  train_loss  valid_loss  error_rate  
3 | 1      0.602823    0.119616    0.049645    (00:14)
```

epoch次数太多 [50:30]

epoch次数太多会导致“过拟合”。如果你训练太久，它会过分地学习识别你提供的那些泰迪而不是普遍的泰迪。这就是过拟合。不管你听说过什么，深度学习里很难出现过拟合。今天我们尝试演示一个过拟合的例子，我关掉所有的优化策略，关掉数据增强（data augmentation）、随机失活（dropout）、权重衰减（weight decay）。尽可能的实现过拟合。使用较小的学习率，训练很长时

间。或许能做到过拟合。仅仅是或许。

识别过拟合的唯一现象是错误率改善了一会后，就变差了。可能很多人，甚至那些声称理解机器学习的人告诉你如果训练损失度比验证损失度小，就是过拟合。今天以及以后你会学到更多细节，这是完全不对的。

所有被正确训练的模型的训练损失度都会比验证损失度小。

这不是过拟合的标志。这不是出错的标志。这是你做对了的标志。过拟合的标志是你的错误率变差了，这是你应该关心的。你希望模型的错误率更低。只要训练时你的模型错误率在变好，就不是过拟合。怎样做到呢？

```
1 np.random.seed(42)
2 data = ImageDataBunch.from_folder(path, train=".", valid_pct=0.9, bs=32,
3         ds_tfms=get_transforms(do_flip=False, max_rotate=0, max_zoom=1,
4         max_lighting=0, max_warp=0
5             ),size=224,
6         num_workers=4).normalize(imagenet_stats)
```

```
1 learn = create_cnn(data, models.resnet50, metrics=error_rate, ps=0, wd=0)
2 learn.unfreeze()
```

```
1 learn.fit_one_cycle(40, slice(1e-6,1e-4))
```

```
1 Total time: 06:39
2 epoch  train_loss  valid_loss  error_rate
3 1      1.513021   1.041628   0.507326   (00:13)
4 2      1.290093   0.994758   0.443223   (00:09)
5 3      1.185764   0.936145   0.410256   (00:09)
6 4      1.117229   0.838402   0.322344   (00:09)
7 5      1.022635   0.734872   0.252747   (00:09)
8 6      0.951374   0.627288   0.192308   (00:10)
9 7      0.916111   0.558621   0.184982   (00:09)
10 8     0.839068   0.503755   0.177656   (00:09)
11 9     0.749610   0.433475   0.144689   (00:09)
12 10    0.678583   0.367560   0.124542   (00:09)
13 11    0.615280   0.327029   0.100733   (00:10)
14 12    0.558776   0.298989   0.095238   (00:09)
15 13    0.518109   0.266998   0.084249   (00:09)
16 14    0.476290   0.257858   0.084249   (00:09)
17 15    0.436865   0.227299   0.067766   (00:09)
18 16    0.457189   0.236593   0.078755   (00:10)
19 17    0.420905   0.240185   0.080586   (00:10)
20 18    0.395686   0.255465   0.082418   (00:09)
21 19    0.373232   0.263469   0.080586   (00:09)
22 20    0.348988   0.258300   0.080586   (00:10)
23 21    0.324616   0.261346   0.080586   (00:09)
24 22    0.311310   0.236431   0.071429   (00:09)
25 23    0.328342   0.245841   0.069597   (00:10)
26 24    0.306411   0.235111   0.064103   (00:10)
27 25    0.289134   0.227465   0.069597   (00:09)
28 26    0.284814   0.226022   0.064103   (00:09)
29 27    0.268398   0.222791   0.067766   (00:09)
30 28    0.255431   0.227751   0.073260   (00:10)
31 29    0.240742   0.235949   0.071429   (00:09)
```

32	30	0.227140	0.225221	0.075092	(00:09)
33	31	0.213877	0.214789	0.069597	(00:09)
34	32	0.201631	0.209382	0.062271	(00:10)
35	33	0.189988	0.210684	0.065934	(00:09)
36	34	0.181293	0.214666	0.073260	(00:09)
37	35	0.184095	0.222575	0.073260	(00:09)
38	36	0.194615	0.229198	0.076923	(00:10)
39	37	0.186165	0.218206	0.075092	(00:09)
40	38	0.176623	0.207198	0.062271	(00:10)
41	39	0.166854	0.207256	0.065934	(00:10)
42	40	0.162692	0.206044	0.062271	(00:09)

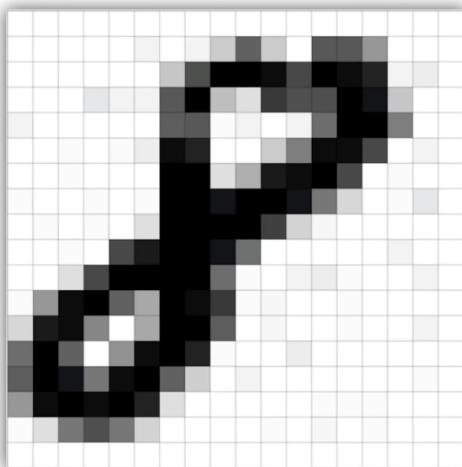
[52:23]

这些是可能引起错误的四种情况。这里有些细节，我们在后面的课程里也会讲到。事实上，如果你现在就不听了，去做实验也没什么问题。去下载图片，用resnet32或者resnet50创建卷积网络，保证学习率和epoch次数正确，部署成Starlette web API，基本上你能做到这些。至少在计算机视觉上没有问题。希望你不要走远，你还要学习自然语言处理、协同过滤、表格数据、图像分割等等。

[53:10]

现在让我们理解下究竟发生了什么。“损失度”是什么意思？“迭代”是什么意思？“学习率”是什么意思？要想理解这些概念，需要知道程序是怎样运行的。我们要学习深度学习的另外一面（理论部分），不再创建一个顶尖水平的美洲豹识别器，我们要回过头来创建一个最简单的线性模型。我们会用到一些数学。不要急着关掉。这没关系。我们会用到一些数学，但是并不难。即使你不喜欢、不擅长数学。因为我们首先要学习的是看像数字8这样的图片。

PREDICTORS ARE FUNCTIONS OF PIXEL VALUES



Number	Prob
0	0.01
1	0.03
2	0.03
3	0.04
4	0.07
5	0.08
6	0.10
7	0.02
np.argmax	8
9	0.01

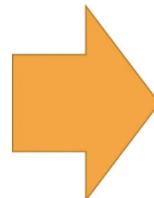


Adam Geitgey [Follow](#)

Interested in computers and machine learning. Likes to write about it.
Jun 13, 2016 · 15 min read

它事实上是一组数字。对于黑白图片来说，它是一个数字矩阵。对于彩色图片来说，它会有三个维度。当你添加一个维度后，我们叫它张量，不再叫它矩阵。它是一个三维数字张量——代表红，绿，蓝。

PREDICTORS ARE FUNCTIONS OF PIXEL VALUES



Number	Prob
0	0.01
1	0.03
2	0.03
3	0.04
4	0.07
5	0.08
6	0.10
7	0.02
$np.argmax$	8
9	0.01



Adam Geitgey

Interested in computers and machine learning. Likes to write about it.

Jun 13, 2016 · 15 min read

我们创建的泰迪熊识别程序实际上可以看成是一个数学函数，它接受泰迪熊图片的数字，把这些数字转换成三个数字：它是泰迪的概率、是灰熊的概率、是黑熊的概率。在这个例子里，有个假想的函数，接收代表手写数字的像素值，返回十个数字：每一个可能结果（0到9这10个数字）的概率。

你 在 我 们 的 代 码 或 者 其 他 深 度 学 习 代 码 里 会 经 常 看 到 这 样 一 组 概 率，还 有 一 个 叫 `max` 或 `argmax` 的 函 数 跟 着。这 个 函 数 是 用 来 找 出 最 高 的 数 值（概 率），告 知 我 们 它 的 索 引 是 什 么。上 面 数 组 的 `np.argmax` 或 者 `torch.argmax` 回 顾 索 引 8。

我们来试一下。我们知道做预测的方法是 `learn.predict`。我们可以在它前面或者后面敲两个问号来查看源代码。

这就是它的源代码。`pred_max = res.argmax()`。它代表什么类别？我们把它传入类别数组`lf.data.classes`就好。`fastai`库里的代码可以增强你对概念的理解，让你知道程序是怎样运行的，有用。

```
Signature: learn.predict(img:fastai.vision.image.Image)
Source:
    def predict(self, img:Image):
        "Return prect class, label and probabilities for `img`."
        ds = self.data.valid_ds
        ds.set_item(img)
        res = self.pred_batch()
        ds.clear_item()
        pred_max = res.argmax()
        return self.data.classes[pred_max],pred_max,res
File:      /data1/jhoward/git/fastai/fastai/vision/learner.py
Type:     method
```

Question: 可以讲下错误率的定义吗，它是怎样算出来的？我觉得它是交叉验证错误。 [56:38]

当然可以。回答“错误率是怎样算出来的”这个问题的一个方法是，输入error_rate??，来看看源代码，代码里写的是 $1 - \text{accuracy}$ （准确率）

```

In [19]: error_rate??
```

```

Signature: error_rate(input:torch.Tensor, targs:torch.Tensor) -> <function NewType.<locals>.new_type at 0x7fdf944d11e0>
Source:
def error_rate(input:Tensor, targs:Tensor)->Rank0Tensor:
    "1 - `accuracy`"
    return 1-accuracy(input, targs)
File:      /data1/jhoward/git/fastai/fastai/metrics.py
Type:     function

```

现在的问题变成了是什么是准确率：

```

Signature: accuracy(input:torch.Tensor, targs:torch.Tensor) -> <function NewType.<locals>.new_type at 0x7fdf944d11e0>
Source:
def accuracy(input:Tensor, targs:Tensor)->Rank0Tensor:
    "Compute accuracy with `targs` when `input` is bs * n_classes."
    n = targs.shape[0]
    input = input.argmax(dim=1).view(n,-1)
    targs = targs.view(n,-1)
    return (input==targs).float().mean()
File:      /data1/jhoward/git/fastai/fastai/metrics.py
Type:     function

```

它是`argmax`。现在我们知道，它是拿预测结果和目标值比较，看有多少次是一致的，取平均值。这就是它的含义。接下来的问题是，在fastai里，它被用在什么地方。这个指标（metrics）经常被用在验证集上。每加入一个指标，它都会被用在验证集上，这是最佳实践。

```

In [ ]: learn = create_cnn(data, models.resnet50, metrics=error_rate, ps=0, wd=0)
learn.unfreeze()

In [ ]: learn.fit_one_cycle(40, slice(1e-6,1e-4))

Total time: 06:39
epoch  train_loss  valid_loss  error_rate
1      1.513021   1.041628   0.507326   (00:13)
2      1.290093   0.994758   0.443223   (00:09)
3      1.185764   0.936145   0.410256   (00:09)
...
```

必须要用模型没有见过的数据来检查它的效果。我们稍后会学习验证集。

有时可能你需要的不是看源代码，你需要的是看文档，你可以输入`doc`。它会列出输入输出的类型，给出一个到完整文档的链接。在文档里你可以找到指标是怎样起作用的，还有什么其他的指标，等等。一般来说，你可以找到更多信息的链接，比如，演示怎样使用这些的示例代码。不要忘记`doc`是你的好帮手。在`doc`方法和文档里，你可以看到一个源码链接。和`??`类似，但是这个链接可以跳转到Github上的具体代码，你可以看到它是怎样实现的，可以看到和它相关的方法。这里有很多有用的东西。

提问：之前为什么使用`3e-3`做学习率？为什么使用`3e-5, 3e-4` [59:11]？

我们发现`3e-3`是一个很好的默认学习率。在解冻前的初始微调中，大部分情况下它都很有效。之后，我倾向于在这个基础上乘以一个倍数。在下一阶段，我选择在这个基础上乘以`0.1`做为`slice`的第二个参数，无论学习率搜索程序找到的一个参数是什么。第二个参数不是从学习率搜索程序得到的。在第一阶段的默认值`3e-3`的基础上乘以`0.1`只是一个经验法则。`slice`的第一个参数是来自于学习率搜索程序的。今天和以后的课程里我们会学习很多关于学习率的细节。现在你只要记住这样的方式就行：

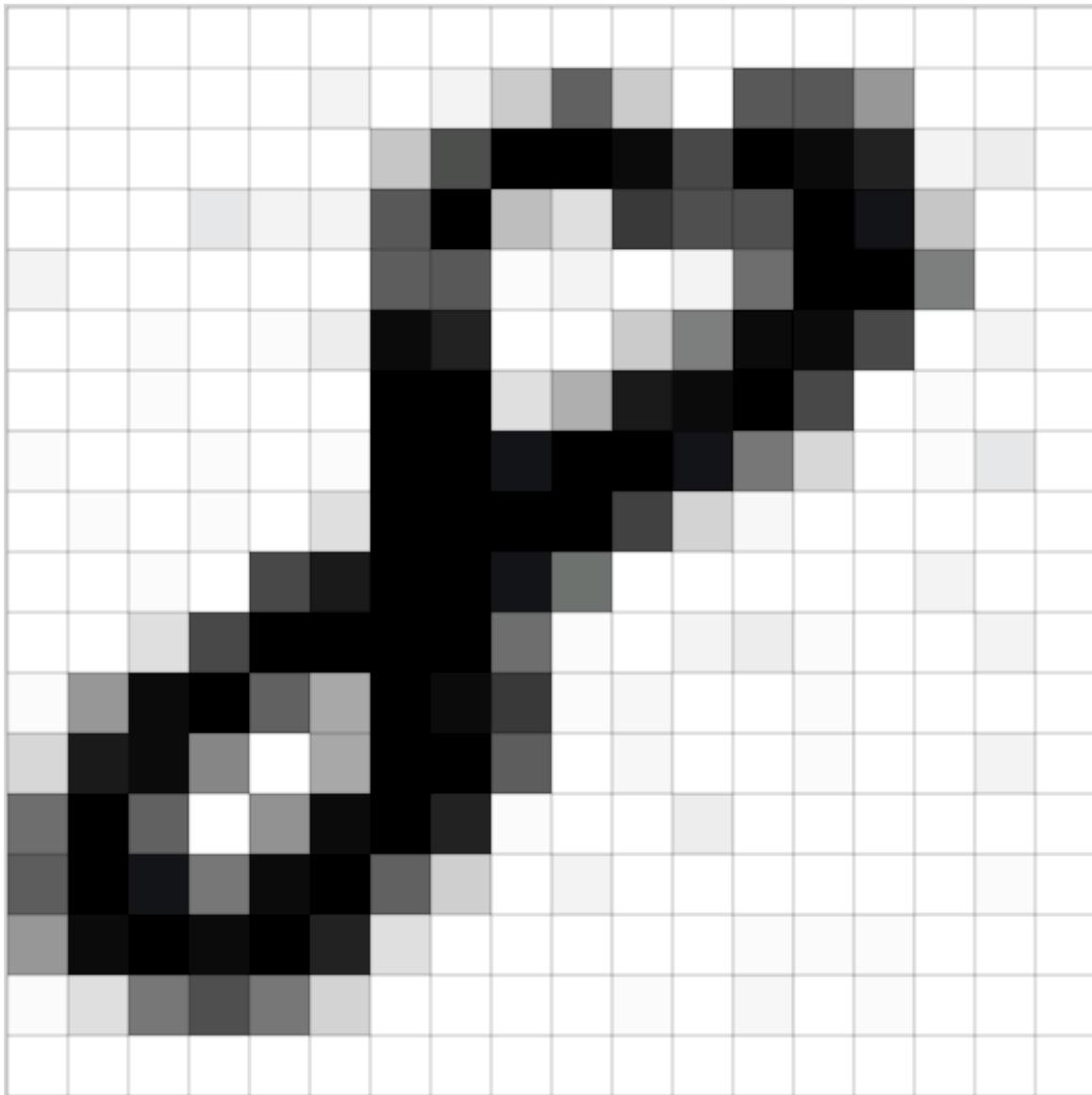
- `learn.fit_one_cycle`
 - 一定数量的epochs，我经常用4
 - 学习率，默认是`3e-3`。我把它显式写出来，这样你能看到它的值
- 运行一会儿，然后解冻它
- 再让它学习一次，这次的学习率在原来的基础上除以10。这次要用`slice`，要给`slice`输入一个参数，这个参数是从学习率搜索程序里得到的，用的是最陡的斜坡的位置的值

```
1 | learn.fit_one_cycle(4, 3e-3)
2 | learn.unfreeze()
3 | learn.fit_one_cycle(4, slice(xxx, 3e-4))
```

这是一个有效的经验法则，不需要考虑太多，不需要知道为什么。但是现在我们深入一些，来进一步理解它。

深入理解数学原理 [1:01:17]

我们来创建这样一个数学函数，输入是表像素的数字，输出是每个可能类别的概率。



顺便讲下，我们这里用的很多东西，是从别人那里拿来的，我们在这里给出关于这些材料的更多信息。请去看下他们的作品，他们做了非常棒的工作，我们在我们的课成里着重展示出来。我很喜欢这个数字动图的主意，感谢 [Adam Geitgey](#) 做了这个。

[1:02:05]

我们看下怎样创建一个这样的函数，我们从一个最简单的函数开始：

$$y = ax + b$$

在这个直线里

- a: 直线斜率
- b: 直线的截距

学习这个课程需要你掌握高中数学，但愿我们讲这个时，你还记得这些，我们假设你还记得。如果我们提到了一些数学里的东西，我认为你还记得，但实际上你已经忘了，不要慌。我们都会这样。[可汗学院](#)很棒。它不单单是给学校孩子们学习的地方。你们去可汗学院，找到你需要重新记起的概念，他解释得很好。强烈推荐。我是一个哲学系学生，我总是会提醒自己，回忆一些知识，也总是会觉得我什么都没学过。我们可以在整个互联网上学习这些。

我重写下这个：

$$y = a_1x + a_2$$

我们只是把 b 换成 a_2 ，换了一个名字。对同一个东西有不同的说法。再换一种写法，我可以写成 a_2 乘以 1。

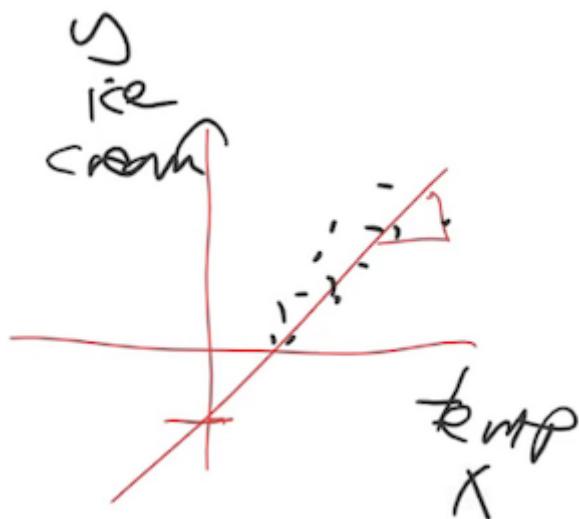
$$y = a_1x + a_2 \cdot 1$$

这还是一回事。现在，我不再用 1，而是在前面用 x_1 ，在后面用 x_2 ：

$$y = a_1x_1 + a_2x_2$$

$$x_2 = 1$$

目前，这些都是高中里低年级的数学知识。我觉得我们可以理解。这和 $y = ax + b$ 是一样的，只是换了个名字。在做机器学习时，我们不是只有一个等式，我们会有很多个。如果我们用代表气温的数据和冰淇淋销量的数据，我们会有很多点。



对于每个点，我们假设它遵循这个公式 ($y = a_1x_1 + a_2x_2$)。这里有很多 y 和很多 x ，我们可以加上一个 i ：

$$y_i = a_1x_{1i} + a_2x_{2i}$$

我们做这个的方式很像 numpy 索引，只是这里我们放在方括号里，而是放在等式的下标里：

⚠ Invalid Equation

这里有很多不同的 y_i ，它们取决于很多不同的 ⚠ Invalid Equation 和 ⚠ Invalid Equation，要注意还有 (a_1, a_2) 。这叫做系数或者参数。这是我们的线性等式，我们还要定义每一个 ⚠ Invalid Equation 都等于 1。为什么要这样做？因为我们要做线性代数。为什么要做线性代数？一个原因是 [Rachel 开了世界上最好的线性代数课程](#)，如果有兴趣你可以学习它。这是一个很好的机会来推广它，虽然我们不会因为这个挣到钱，但这没关系。更重要的是，它会让事情更容易。我讨厌写循环，讨厌写代码，只想让电脑为我做所有的事。只要你看到这个 i 的下标，这意味着你要做循环和各种事情了。你大概记得把两个数相乘，另两个数相乘，然后加在一起，这叫做“点积 (dot product)”。如果你为很

多个有不同下标的数做同样的事，这叫做矩阵积（matrix product）。事实上，整个的这些可以写成这样：

$$\vec{y} = X\vec{a}$$

不再是很多个不同的 y_i ，我们可以说这里有一个叫做 \vec{y} 的向量，它等于一个叫做 X 的矩阵乘以一个叫做 \vec{a} 的向量。我知道你们当中的很多人不记得这些了。没关系，有一个图可以展示这个。

Andre Stoltz创建了这个神奇的网站 <http://matrixmultiplication.xyz/>，现在我们有了一个矩阵和一个向量，我们可以得到一个乘积。

Matrix Multiplication

$$\begin{array}{c} \begin{bmatrix} - \\ + \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 0 \\ 2 & 3 & 4 \end{bmatrix} \times \begin{bmatrix} 2 \\ 6 \\ 1 \end{bmatrix} \quad \begin{bmatrix} - \\ + \end{bmatrix} \\ \begin{bmatrix} - & + \end{bmatrix} \quad \begin{bmatrix} - & + \end{bmatrix} \end{array}$$

 Multiply

这就是矩阵向量乘法。换种说法，它就是  Invalid Equation，不过他的版本更容易理解。

提问: 创建新的数据集时，怎样知道多少图片是足够的？怎样衡量是否足够？ [\[1:08:35\]](#)

很好的问题。另外一个可能的问题是你没有足够的数据。怎样知道你没有足够的数据呢？你找到了一个好的学习率（如果你把学习率调大，会导致很大的损失度；如果你把它调小，它会变得很慢），你训练很长时间，你的错误率开始变差。这样你知道你训练了足够久。你对准确率还是不满意——对你想要的泰迪拥抱安全级别还不够。如果发生了这样的情况，有几种方法来处理，我们会在这个课程里学习这些方法。但是其中最简单的就是获取更多数据。如果你获取更多数据，然后你可以训练更长时间，得到一个更高的准确率，更低的错误率，也不会出现过拟合。

我也希望有简单的方法，可惜没有。我也希望有能提前知道需要多少数据的方法。但是我可以说大部分情况下，你需要的数据比你认为的要少。很多组织经常花了太多时间来获取数据，得到的数据多于他们真正需要的。所以你可以先不用准备那么多数据，然后看看是否足够。

Question: 如果数据不均衡会怎样？比如说200张灰熊图片和50张泰迪图片 [\[1:10:00\]](#)

没关系。试一下。它照样可以用。很多人问我怎样处理不均衡的数据。我过去几年用不均衡的数据做了很多分析，这不会让模型失效。有一篇论文说如果你希望优化模型，最好的方法是取一个不常用的分类，复制几份。这叫“过度样本”。实践中我没有遇到需要这样做的情况。我认为对我来说它一直可以正常运行。

提问：当你解冻完，重新训练一轮时，如果训练集损失度仍然比验证集损失度高（没有拟合），你会重新训练一遍没有解冻的模型（实现上，可能会多于一轮），还是会用更多的epoch重新整个训练一遍？ [\[1:10:47\]](#)

你们上周问过我这个问题。我的答案还是一样的。我不知道哪个更好，两个都可以。如果你再做一轮，结果可能会变好些。如果你重新开始，做两遍会花比较久，这有些烦人，取决于你的耐心。这不会有太大差别。对我个人来说，我一般会多训练几轮。但是大多数情况下，两种方式没什么差别。

Question: 关于这个示例代码：

```
1 classes = ['black', 'grizzly', 'teddys']
2 data2 = ImageDataBunch.single_from_classes(path, classes,
3     tfms=get_transforms(), size=224).normalize(imagenet_stats)
4 learn = create_cnn(data2, models.resnet34)
5 learn.load('stage-2')
```

我对 `models.resnet34` 这里感到奇怪：我的理解是这个模型是用 `.save(...)` 方法保存的数据（在硬盘上大概85MB）创建的，不再需要一个 `resnet34` 的拷贝（a copy of `resnet34`）。 [\[1:11:37\]](#)

稍后我们会学习这个。没有“ResNet34的副本（a copy of `resnet34`）”这种东西，ResNet34 是一种架构——是一种函数形式。就像 $y = ax + b$ 是一个线性函数形式。它不占用任何空间，不包含任何东西，它只是一个函数。ResNet34只是一个函数。我想引起混淆是因为我们经常使用从ImageNet学习到的预训练模型。这里我们不需要用预训练模型。事实上，要想不使用它，你可以传入 `pretrained=False`：

```
learn = create_cnn(data, models.resnet34, metrics=error_rate, pretrained=False)
```

这会保证不加载预训练模型，我猜这会节省0.2秒。我们会学习很多关于这个的内容。如果还不太清楚的话也不要担心。`models.resnet34` 只是说它是一个线性函数还是一个二次函数，还是一个倒数——它只是一个函数。这是一个ResNet34函数。它是一个数学函数，不占用任何存储空间，不包含数据，不需要被加载，而预训练模型需要被加载。我们做推理时，这个会占用空间：



```
In [ ]: classes = ['black', 'grizzly', 'teddys']
data2 = ImageDataBunch.single_from_classes(path, classes, tfms=get_transforms(), size=224).normalize(imagenet_stats)
learn = create_cnn(data2, models.resnet34)
learn.load('stage-2')
```

在这里我们加载参数。这些参数，像之前讲的那样，是 a 和 b 的值——我们需要保存这些数字。对于ResNet 34，你不是只保存两个数字，你需要保存几百万或者几千万个数字。

[\[1:14:13\]](#)

我们为什么做这些？因为我希望能把它写成这样： $\vec{y} = X\vec{a}$ ，写成这样的原因是这样就可以在PyTorch里处理，不需要循环，一行代码就可以，也会执行地更快。

它希望你输入一个等式来一次性计算。这意味着，你应该试着用这种线性代数的形式定义问题。我们要尝试用的是这个 $\vec{y} = X\vec{a}$ （我们把它叫一种架构）。这是世界上最小的神经网络。它有两个参数 a_1 和 a_2 。我们来尝试用一些数据拟合这个架构。

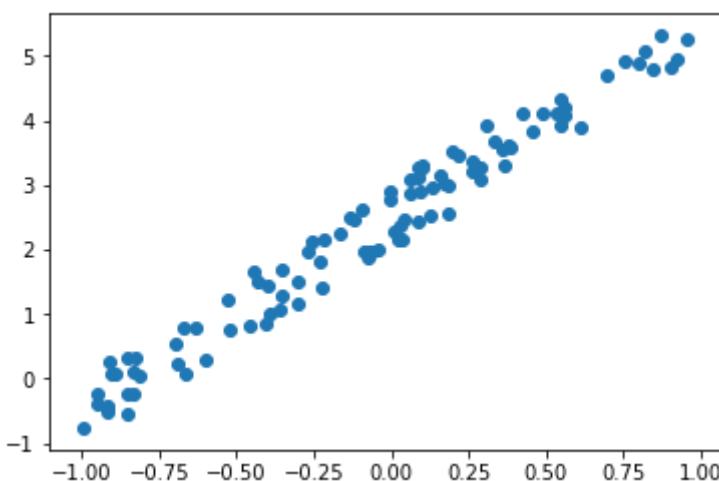
随机梯度下降 [1:15:06]

我们来打开一个notebook，生成一些点，看看能不能用一些方法让它拟合一个直线。我们要用的方法被叫做SGD。什么是SGD？有两种SGD。第一个是我在第一课说“你们应该都试着构建这些模型，试着做些酷的东西”，你们都做了实验，做出了很好的东西。这里S代表Student。这里SGD是Student Gradient Descent。这是第一个版本的SGD。

第二个版本的SGD是我们今天要讲的，我们要用电脑尝试很多东西来找到一个好的函数，这被叫做 Stochastic Gradient Descent（随机梯度下降）。另外一个你经常在Twitter上听到的是Stochastic Grad student Descent。

线性回归问题 [1:16:08]

我们打开[lesson2-sgd.ipynb](#)。我们现在要从下到上（从细节到整体）地学，不再是从上到下。我们要创建最简单的模型，这是一个线性模型。首先我们需要一些数据。我们要生成一些数据。我们要生成的数据看起来是这样的：



x轴大概代表气温，y轴大概代表冰淇淋销量或者类似的东西。我们要创建一些线性的模拟数据。在创建这个时，我们也会学习一些PyTorch的知识。

```
1 | %matplotlib inline
2 | from fastai import *
```

```
1 | n=100
```

```
1 | x = torch.ones(n,2)
2 | x[:,0].uniform_(-1.,1)
3 | x[:5]
```

```
1 | tensor([[-0.1338,  1.0000],
2 |          [-0.4062,  1.0000],
3 |          [-0.3621,  1.0000],
4 |          [ 0.4551,  1.0000],
5 |          [-0.8161,  1.0000]]))
```

我们生成这些数据的方式是创建一些系数。 a_1 设成3, a_2 设成2。我们要为 x 创建一列，里面数字全部是1。

```
1 | a = tensor(3., 2); a
```

```
1 | tensor([3., 2.])
```

然后我们做这个 $x@a$ 。什么是 $x@a$? $x@a$ 在Python里表示 x 和 a 的矩阵乘积。事实上，它有更广泛的含义。它可以是向量向量乘积、矩阵向量乘积、向量矩阵乘积、或者矩阵矩阵乘积。在PyTorch里，当我们处理高秩（高维）张量时，它可以代表更多东西，我们很快就会学到。这基本上是在所有的深度学习里都要做的一件重要的事情。绝大多数时候，电脑会这样把数字相乘然后相加，这是很有用的操作。

```
1 | y = x@a + torch.rand(n)
```

[1:17:57]

我们这样来生成数据：创建一个线性函数，然后再输入一些随机数。让我们回过头来看看怎样创建 x 和 a 。我讲过我有这两个系数3和2。你可以看到它们被存在这个叫做 `tensor` 的函数里。你之前可能听说过“`tensor`（张量）”这个单词。这种词听起来有些吓人，如果你是一个物理学家可能听起来觉得很平常。但是在深度学习里，它一点也不吓人。它代表数组（array），特别指形状规则（regular shape）的数组。不是那种第一行有2个元素、第三行有3个元素、第4行有1个元素的数组，这被叫做“交错数组（jagged array）”。这不是张量。张量的形状是矩形或者立方体或者每行每列的长度都相等。下面这些都是张量：

- 4×3 矩阵
- 长度是4的向量
- $3 \times 4 \times 6$ 的3维数组

这都是张量。我们一直在用这些。比如一个图片是一个3维张量。它的形状是行数 \times 列数 \times 通道数（通常是红，绿，蓝）。通常，VGA图片是 $640 \times 480 \times 3$ ，这些是在底层做的。当人们谈论图片时，通常用宽和高，当我们用数学来描述时，我们使用行数，列数，所以它是 $640 \times 480 \times 3$ ，这让你难以理解。对张量来说，我们不用维度（dimensions）。我们用这两个词，秩（rank）或者轴（axis）。秩代表有多少个轴，多少个维度。所以一个图片一般是一个秩为3的张量。我们这里创建的是一个秩为1的张量（也就是向量）。在数学里，人们用不同的名字来表示有些不同的概念。一维数组叫向量，二维数组叫矩阵，三维数组没有名字。这没什么道理。在计算机里，我们尝试使用简单一致的命名。都叫张量——秩为1的张量、秩为2的张量、秩为3的张量。你可以创建秩为4的张量。如果你有64张图片，这就是一个秩为4的 $64 \times 480 \times 640 \times 3$ 的张量。所以张量很简单。它们就是数组。

在PyTorch里，你输入 `tensor`，传入一些数字，在这个例子里，得到一个列表，一个向量。这代表我们的系数：直线的斜率和截距。

```
a = tensor(3., 2); a
```

```
: tensor([3., 2.])
```

我们不会用这种特殊的 $ax + b$ ，我们第二个 x 的值设成1



这里可以看到，把值设成1可以让我们做简单的矩阵向量乘积：

```
x = torch.ones(n, 2)
x[:, 0].uniform_(-1., 1)
x[:5]

tensor([[-0.1338,  1.0000],
       [-0.4062,  1.0000],
       [-0.3621,  1.0000],
       [ 0.4551,  1.0000],
       [-0.8161,  1.0000]])
```

这就是 a 。然后我们需要生成这个 x 数组。我们要向第一列放入随机数，向第二列放入1。要做到这个，我们用PyTorch创建一个秩是2的张量，格式是 $n \times 2$ 。我们传入了两个参数，得到一个的秩是2的张量。行数是 n ，列数是2。在里面，所有的数都是1——这就是`torch.ones`的含义。

[1:22:45]

下面这个很重要。你可以用Python里对list做索引的同样的方法，来对张量做索引。你可以在任何地方放一个冒号，冒号代表这个轴/维度的每一个数。这里的`x[:, 0]`代表的第0列的每一个行。所以`x[:, 0].uniform_(-1., 1)`是让第0列的每一行，成为服从均匀分布的随机数（uniform random numbers）。

还有一个PyTorch里的重要概念。每次你看到一个以下划线结尾的函数，它代表不要返回随机数，而是把函数的结果赋值给调用这个方法的变量。所以这个`x[:, 0].uniform_(-1., 1)`把第0列替换成-1到1之间的服从均匀分布的随机数。这里有很多内容要学。

```
x = torch.ones(n, 2)
x[:, 0].uniform_(-1., 1)
x[:5]

tensor([[-0.1338,  1.0000],
       [-0.4062,  1.0000],
       [-0.3621,  1.0000],
       [ 0.4551,  1.0000],
       [-0.8161,  1.0000]])
```

好消息是这两行代码和我们就要用到的`x@a`覆盖了PyTorch里你需要知道的95%的知识。

1. 怎样创建数组
2. 怎样修改数组内容
3. 怎样对数组做向量操作

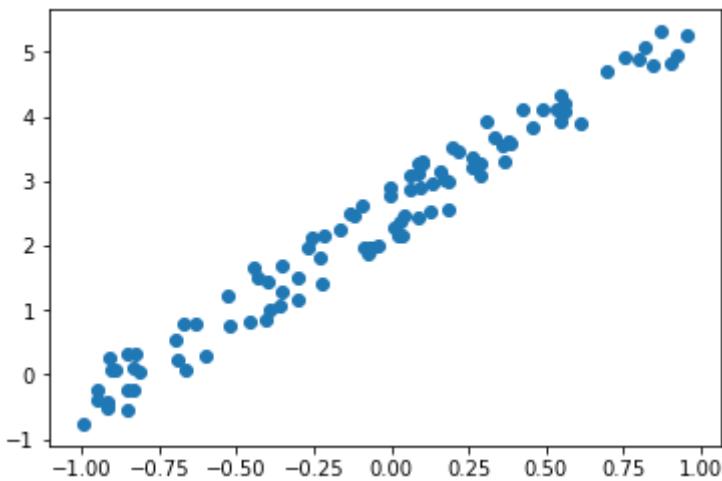
有很多内容要学，但这几个概念是极其重要、极其有用的。现在我可以打印出前5行。`[:5]`是一个标准的Python分割符号，代表前5行。这是前5行，总共两列：一列随机数，一列1。

现在我可以对`x`和`a`做乘积，添加一个随机数做噪音。

```
1 | y = x@a + torch.rand(n)
```

然后我画一个散点图。我不关心全部是1的那一列，它只是用来让线性函数更方便。所以我只画第0列和`y`的散点图。

```
1 | plt.scatter(x[:,0], y);
```



`plt`一般用来代表画图库matplotlib。大部分人在大多数Python科学计算里使用它来画图。这是你应该掌握的一个库，把数据画出来是相当重要的。还有很多其他的画图库，这些库在某些方面比matplotlib好，但matplotlib在所有方面的表现都比较均衡。有时会有点不太好用，但是我基本都是用matplotlib，因为没有什么功能是它做不到的，尽管有些库在某些方面会稍微好些。它确实很强大，你学会了matplotlib，你就可以做所有事情。这里，我使用matplotlib来画x和y的散点图。这是我的气温和冰淇淋销量的模拟数据。

[1:26:18]

现在我们要做的是，假装拿到了这些数据，但不知道3和2这两个系数。假装从来不知道这个系数的值，要把它们找出来。怎样找出来？怎样画一条线来拟合这些数据？这有什么意义？关于它有什么意义我们稍后会讲更多。基本上是这样：

如果我们可以找出可以拟合这100个点的函数的两个参数，我们也可以拟合把像素值转成概率的任意函数。

我们要学习的这个能找出这两个数的技术，同样可以用来找ResNet34里的5千万个参数。我们会用几乎一样的方式。在过去的班级里我发现有些人很难理解这一点。我经常遇到，甚至在第四五周之后，有人来问我：

学生：我不太明白，我们究竟是怎样训练这些模型的？

Jeremy：用的是SGD。就是找两个系数的notebook里用的那个。

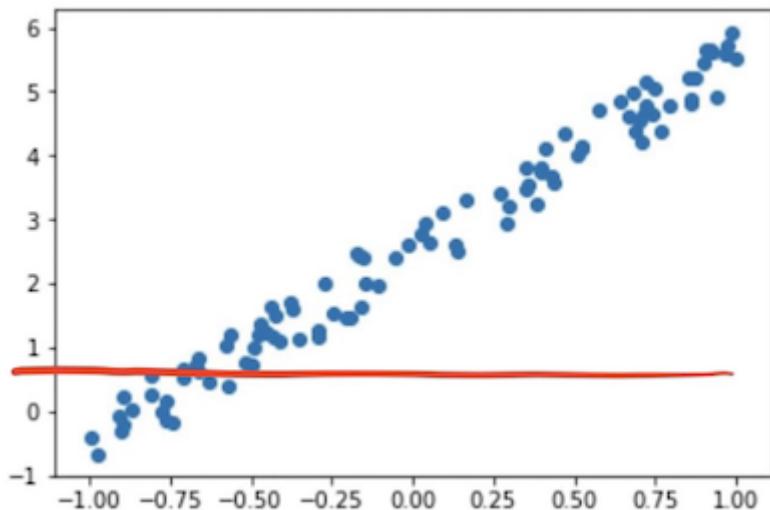
学生：但是我们不是要拟合一个神经网络吗？

Jeremy：是的。但我们没办法打印出5千万个数字，不过它们做的事情是完全一样的

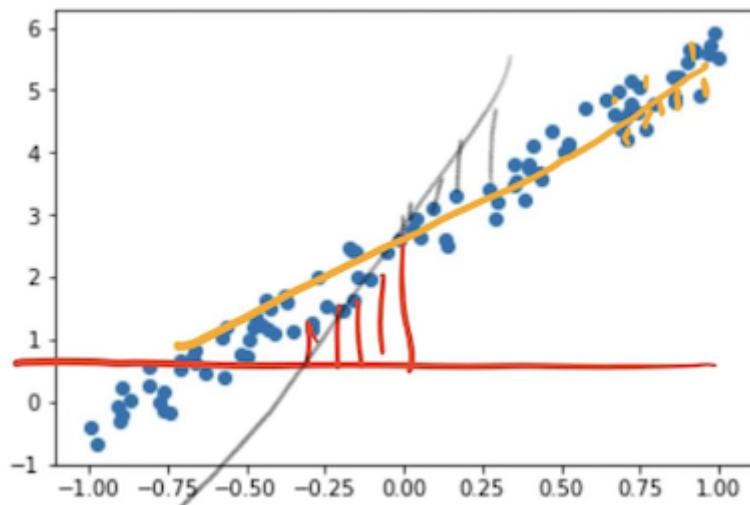
难以理解这个的原因是人类的大脑很难建立“找5千万个参数和我们找2个参数一样”这样的概念。现在，你要记住我说的。它可以做识别泰迪熊这样的事。这些函数可以很强大。我们会学习怎样让它们超级强大。现在，我们要学的这个拟合两个数的方法和拟合5千万个数的方法是一样的。

损失函数 [1:28:36]

我们要找出PyTorch里叫**参数 (parameters)** 的东西（比如 a_1 和 a_2 ），在统计学里，它们被叫做系数 (coefficient)。我要找的系数要能够让它们对应的线到这些点之间的差距最小。换句话说，如果我们找到的 a_1 和 a_2 画出来的线是这样：



我们可以看到这条线到每个点之间的距离。这差了很多。应该有其他的 a_1 和 a_2 能画出这条灰线。看下现在的距离。然后我们看看这条黄线。这回，每个点都很近。



我们可以得到在每个点到线的距离，再对这些距离取平均值。这被叫做**损失**。这是我们的损失值。你需要一个数学函数来计算线到这些点的距离。

对于这种被叫做回归 (regression) 的问题（这里你的因变量 (dependent variable) 是连续的，和选灰熊还是泰迪不同，它是介于-1到6之间的数字），最常用的损失函数被叫做均方差，(mean squared error，缩写MSE)。你可能还见过均方根误差 (root mean squared error，缩写RMSE)。均方差是你的预测值，比如线的值，和冰淇淋销量这个实际值之间的差距。在数学中，人们经常用 y 代表真实值，用 \hat{y} (y hat) 代表预测值。

当写均方差等式之类的东西时，我们不会写“冰淇淋”和“气温”，因为我们希望它能用在所有的东西上。所以我会使用这些数学符号。

均方差只是这些差(y_hat-y)做平方，然后取平均值。因为 y_hat 和 y 都是秩为1的张量，我们可以用一个向量减去另一个向量，它会做“元素运算”，也就是它把每个对应的元素相减，最终我们得到一个差的向量。如果我们对它做平方，它会为里面每个元素做平方。然后我们可以对它取平均值，得到实际值和预测值的差的平方的平均数。

```
1 | def mse(y_hat, y): return ((y_hat-y)**2).mean()
```

如果你熟悉数学符号，我们写的是：

$$\frac{\sum(\hat{y} - y)^2}{n}$$

我要提一下，我并不觉得 `((y_hat-y)**2).mean()` 比 $\frac{\sum(\hat{y} - y)^2}{n}$ 更复杂笨重，代码的优点是，

你可以执行它来实验。只要你定义了它，就可以执行它，传入参数，得到结果，看看它运行的怎样。对我来说，大多数时间，我喜欢用代码来解释，而不是数学公式。它的意思是一样的，只是符号不同。一种符号是可以执行的，你可以用它来实验。另外一种符号是抽象的。这就是为什么我总是用代码来演示。

好消息是，如果你是一个没有太多数学背景的开发者的话，这比较有利，其实你们都有数学背景。因为代码就是数学。如果你的数学背景比较强，代码背景比较弱，你在数学里学的很多东西都可以直接转换成代码，这样你就可以做实验了。

[1:34:03]

`mse` 是一个损失函数。它可以告诉我们这个直线的效果。现在我们需要找到可以拟合的线。记住我们要假装我们不知道。所以你只能去猜。你需要猜出 a_1 和 a_2 的值。假如我们猜 a_1 和 a_2 是 -1 和 1。

```
1 | a = tensor(-1., 1)
```

这是创建张量的方法。我这样写是因为你总是会看到这样的写法。写完整的话，是 `tensor(-1.0, 1.0)`。不能没有点，因为 `tensor(-1, 1)` 是一个整数，不是浮点数。如果你用整数做计算，在神经网络里是会出错的。

我每次都懒得写 `.0`。Python 可以聪明地知道如果你在任何一个数后面加了 `.`，所有的数都是浮点。这就是为什么你总是会看到这样的写法，尤其是像我一样的懒人。

`a` 是一个张量。你可以看到它是一个浮点。你看，PyTorch 也很懒。它只写了一个点，没有写出 0。

```
a  
tensor([-1., 1.])
```

如果你想看到准确的类型，可以写 `.type()`，你可以看到它是一个 `FloatTensor`：

```
a.type()  
'torch.FloatTensor'
```

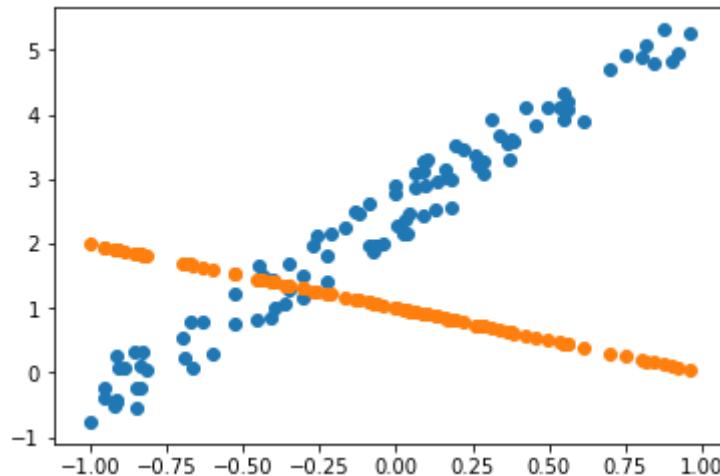
现在我们可以用随机猜的这个数计算预测值，`x@a` 是一个 `x` 和 `a` 的矩阵积。我们现在可以计算预测值和实际值的均方差，这就是损失度。对应这次回归来讲，损失度是 0.9。

```
1 | y_hat = x@a  
2 | mse(y_hat, y)
```

```
1 | tensor(8.8945)
```

现在我们可以画出 `x` 和 `y` 的散点图，可以画出 `x` 和 `y_hat` 的散点图。是这样的：

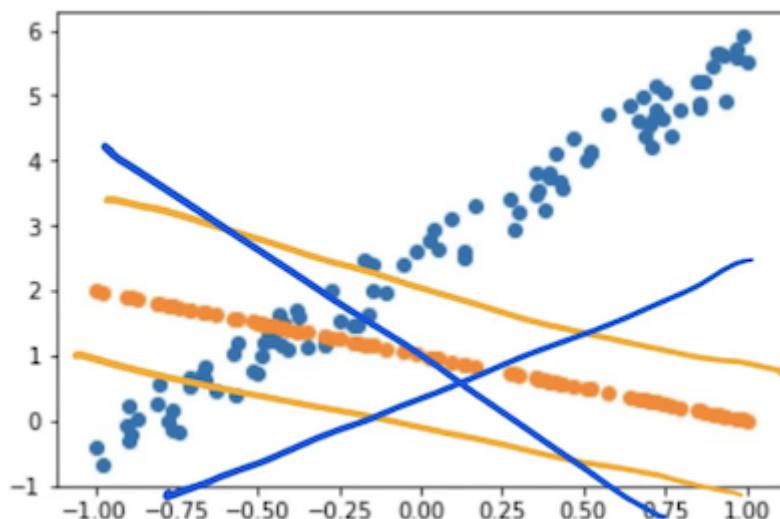
```
1 | plt.scatter(x[:,0],y)
2 | plt.scatter(x[:,0],y_hat);
```



效果并不好，一点儿也不意外。这只是随便猜的值。下面来看SGD（随机梯度下降）或者更通用的梯度下降（gradient descent），在学校里学工程或者计算机的人可能在学校里用牛顿法（Newton's method）做过很多。如果你没有做过，没关系，我们现在就来学习。

它就是猜测和尝试不同的值，来让结果变好一点。怎样变好呢？这里有两个数，它们是橙色线的截距和斜率。我们用梯度下降要做的是：

- 如果我们对这两个数做一小点改变会怎样？
 - 如果我们把截距变大或变小一点会怎样？
 - 如果我们把斜率变大或变小一点会怎样？



有四种可能的组合，我们可以计算这四种情况的损失，看看哪种更好？把它上移还是下移比较好？让它更倾斜还是更加水平比较好？我们要做的就是，看那种方式能让结果更好，我们就按照哪种方式去做。就是这样。

对于记得微积分的人来说，有个好用的方法。你不需要上下移动或者旋转它。你可以计算导数。导数可以告诉你是上移还是下移比较好，这样转还是那样转比较好。好消息是如果你没有做过微积分或者已经忘记了微积分，我可以讲给你所有你需要的关于它的知识。它会告诉你改变一个东西会怎样改变这个函数。某种意义上，导数也被叫做梯度，这不太准确，但足够接近。梯度或者导数会告诉你怎样修改 a_1 来改变MSE，怎样修改 a_2 来改变MSE，这比上下移动来尝试快的多。

在学校里，他们强迫我们坐在哪里手工计算这些导数。我们现在有电脑，电脑可以为我们做这些了，我们不再需要手工计算导数。

```
1 | a = nn.Parameter(a); a
```

```
1 | Parameter containing:  
2 | tensor([-1., 1.], requires_grad=True)
```

[1:39:12]

我们调用 `.backward()` 来做这个。在电脑上，这可以为我们计算梯度。

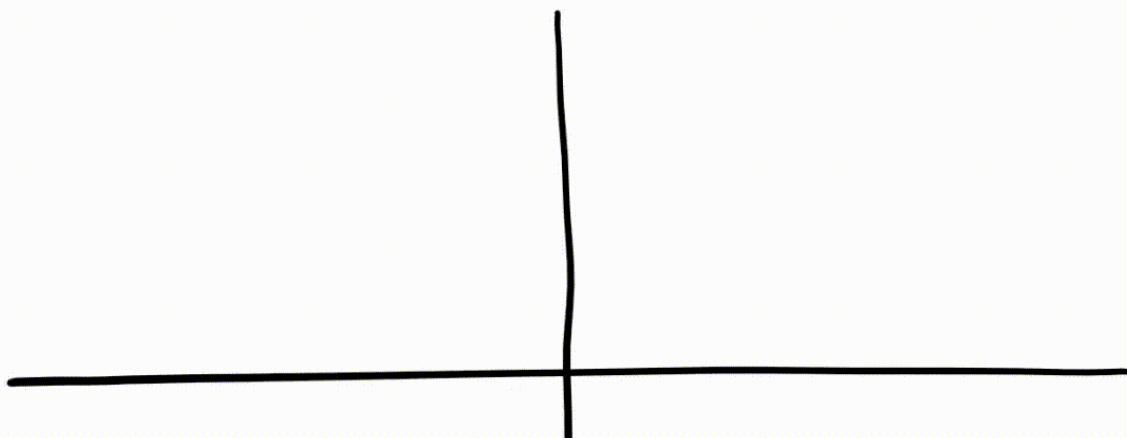
```
1 | def update():  
2 |     y_hat = x@a  
3 |     loss = mse(y, y_hat)  
4 |     if t % 10 == 0: print(loss)  
5 |     loss.backward()  
6 |     with torch.no_grad():  
7 |         a.sub_(lr * a.grad)  
8 |         a.grad.zero_()
```

这就是我们要做的。我们会创建一个循环，循环100次。循环里我们会学会调用一个叫 `update` 的函数。这个函数会：

- 计算 `y_hat` (预测值)
- 计算损失度 (均方差)
- 每一次它都会打印出来，我们就可以看到运行的怎样
- 计算梯度。在PyTorch里，计算gradient是一个叫做 `backward` 的方法做的。均方差只是一个简单的标准数学函数。PyTorch一直追踪它的计算，让我们计算出导数。如果你在PyTorch里做数学计算，你可以调用 `backward` 来计算导数，导数存在一个叫 `.grad` 的属性里。
- 把我的系数减去梯度 (`sub_`)。这里有一个下划线，因为它会原地执行，它会用减去梯度后的值来更新系数 `a`。为什么做减法？因为如果参数沿着梯度的方向改变，损失度会上升。如果朝相反的方向改变，损失度会下降。我们希望损失度最小。这就是为什么做减法。
- `lr` 是我们的学习率。它是一个我们乘到梯度上的东西。为什么要用 `lr`？我来告诉你们。

为什么要用学习率 [1:41:31]

我们看一个简单的例子，一个二次曲线。我们的算法的任务是找到曲线的最低点。怎样做到呢？就像我们在做的那样，随机选择一些x来开始。然后找出y的值。这是起点。然后它可以计算梯度，梯度就是这个斜线，它告诉你哪个方向是向下的。梯度告诉你，你要走这个方向。



- 如果梯度非常大，你可能会向左走很多。你可能会越过到这里。如果你越过了，就没有用了，损失度会变差。我们跳了太远，这不是我们想要的。
- 或许我们应该仅仅跳一小点。这样确实近了一点，然后我们再跳一小步。看看梯度是怎样的，然后再跳一小步，一直重复。
- 换句话说，我们用梯度来告诉我们向哪个方向移动。然后我们用一个小于1的数乘以它，这样才能不会跳得太远。

希望这会让你记起如果学习率太高会发生什么。

Learning rate (LR) too high

```
learn = create_cnn(data, models.resnet34, metrics=error_rate)

learn.fit_one_cycle(1, max_lr=0.5)

Total time: 00:13
epoch  train_loss  valid_loss  error_rate
1      12.220007  1144188288.000000  0.765957    (00:13)
```

现在明白为什么会这样了吧？学习率太高会让我们直接越过正确的值，比起点还要差。这就是学习率太高时发生的情况。

反过来，如果学习率太低，你的每一步都非常小，这样你可以找到正确的值，但是一路上要做很多很多计算。所以你应该找到一个既足够大，也能让你落在正确值附近的学习率。应该让你快速找到，又不能太快以至于越过，不能太慢，走太多步。这就是为什么我们需要一个好的学习率，这就是它的作用。

如果你读任何深度学习库的代码，你都会看到这个：

```
a.sub_(lr * a.grad)
```

你都能找到这种代表“系数-学习率*梯度”的语句。我们会学习一些简单但重要的优化，让它更快些。

有几个小问题，不需要在这里讲太细：一个是把梯度设成0，一个是确保在做SGD更新时关闭梯度计算。如果你有兴趣，我们可以在论坛上讨论这些，你也可以参加我们的机器学习课程，里面有所有这些东西的细节。

训练循环 [1:45:43]

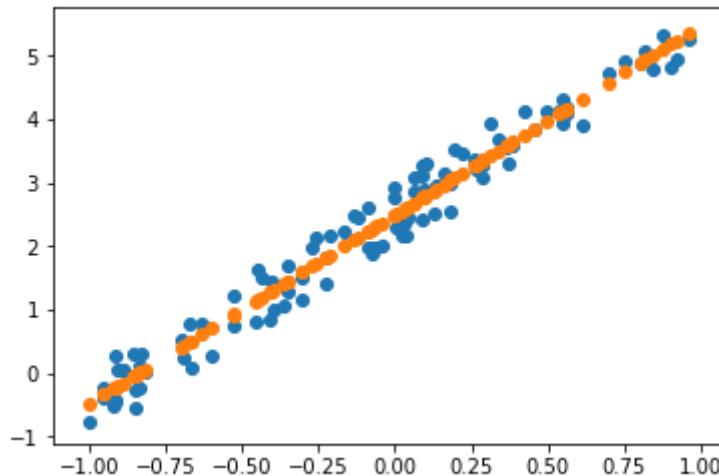
如果我们运行 `update` 100次，间隔几次就打印出损失度，你可以看到它开始是8.9，然后一直变小。

```
1 lr = 1e-1
2 for t in range(100): update()
```

```
1 tensor(8.8945, grad_fn=<MeanBackward1>)
2 tensor(1.6115, grad_fn=<MeanBackward1>)
3 tensor(0.5759, grad_fn=<MeanBackward1>)
4 tensor(0.2435, grad_fn=<MeanBackward1>)
5 tensor(0.1356, grad_fn=<MeanBackward1>)
6 tensor(0.1006, grad_fn=<MeanBackward1>)
7 tensor(0.0892, grad_fn=<MeanBackward1>)
8 tensor(0.0855, grad_fn=<MeanBackward1>)
9 tensor(0.0843, grad_fn=<MeanBackward1>)
10 tensor(0.0839, grad_fn=<MeanBackward1>)
```

然后你可以打印出散点图。

```
1 | plt.scatter(x[:,0],y)
2 | plt.scatter(x[:,0],x@a);
```



拟合成功了。不管你信不信，这就是梯度下降。我们只是用一个比 $x@a$ 稍微复杂的函数作为开始，如果我们有一个可以表示图片是否是泰迪熊的函数，我们也可以用相同的方法来拟合它。

做成动画 [1:46:20]

现在我们看看这个过程的动画。这是一个可以用matplotlib做的很有用的事。你可以画任何东西，把它变成动画。这样你可以看到它每一步的更新。

```
1 | from matplotlib import animation, rc
2 | rc('animation', html='html5')
```

第一次运行时，你可能需要取消掉下面的注释来安装必要的插件：

(运行完下面的命令后，重启notebook的内核)

如果你在colab上运行，不用再安装这些；只需要把上面的... html='html5'改成 ... html='jshtml'

```
1 | #! sudo add-apt-repository -y ppa:mc3man/trusty-media
2 | #! sudo apt-get update -y
3 | #! sudo apt-get install -y ffmpeg
4 | #! sudo apt-get install -y frei0r-plugins
```

来看下我们在这里做了什么。我们只是像以前一样，创建了一个散点图，但是没有用循环，我们用了matplotlib里的 `FuncAnimation` 来调用100次这个 `animate` 函数。这个函数只是调用我们之前创建的 `update`，然后更新直线里的 `y` 的数据。重复100次，每次执行后等待20毫秒。

```
1 | a = nn.Parameter(tensor(-1.,1))
2 |
3 | fig = plt.figure()
4 | plt.scatter(x[:,0], y, c='orange')
5 | line, = plt.plot(x[:,0], x@a)
6 | plt.close()
7 |
8 | def animate(i):
9 |     update()
10 |     line.set_ydata(x@a)
11 |     return line,
12 |
13 | animation.FuncAnimation(fig, animate, np.arange(0, 100), interval=20)
```



你可能会以为用动画可视化你的算法是令人惊奇的并且复杂的事情，事实上，你可以看到，只需要11行代码。所以我认为这非常酷。

这就是可视化的随机梯度下降，我们要可视化更新ResNet34的五千万个参数不会像这个一样方便，但它们本质上是相同的。学习这个简单的版本是一个很好的获得直观感觉的方式。你应该用一个很大的学习率，再用一个很小的学习率来运行下这个notebook，来获得对这个算法的直观的感觉。或许你可以试着做一个三维的图。我没尝试过，但我认为它可以运行得很好。

Mini-batches (小批次) [1:48:08]

这和随机梯度下降的唯一的区别是被叫做*mini-batches*的东西。你可以看到，我们这里做的是我们在每个迭代计算了整个数据集的损失度。如果你的数据集是ImageNet里的150万张图片，这会非常慢。要更新一次参数，你需要计算150万张图片的损失。你不会喜欢这样的情况。我们会一次随机选择大概64张图片，然后计算这64张图片的损失，然后更新我们的权重。换句话说，循环还是一样的，但是每次会对x和y选择一些随机索引来做一个mini-batch，这就是根本的区别。

```
1 def update():
2     y_hat = x[rand_idx]@a
3     loss = mse(y[rand_idx], y_hat)
4     if t % 10 == 0: print(loss)
5     loss.backward()
6     with torch.no_grad():
7         a.sub_(lr * a.grad)
8         a.grad.zero_()
```

如果你每次都选择一些随机数据，这些随机数据被叫做你的mini-batch数据。这种方法被叫做随机梯度下降 (SGD, Stochastic Gradient Descent) 。

术语 [1:49:40]

我们刚刚讲了一些术语，回顾一下。

- **学习率**: 我们乘在梯度上的数，用来决定对权重做多大幅度的更新。
- **Epochs**: 对所有数据（比如所有的图片）的一次完整遍历。对于梯度下降（不是随机梯度下降），在每个循环，我们处理整个数据集。如果你的数据集有1000张图片，我们的小批次尺寸是100，要把所有图片看一遍需要10个迭代。这就是一个Epoch（一轮）。Epochs很重要，如果你做了很多轮，就是你看了图片很多遍，每次你看一遍这个图片，过拟合的可能性就变大了一点。所以我们一般不希望做很多轮。
- **Mini-batch 小批次**: 用来更新权重的一组随机数据。
- **SGD**: 用小批次做随机梯度下降
- **模型 (Model) / 架构 (Architecture)** : 它们代表相同的东西。这个例子里我们的架构是 $\vec{y} = X\vec{a}$ ，架构是给你拟合参数的数学函数。我们晚些会学习像ResNet34这样东西的数学函数是什么。就像你刚刚看到的那样，它基本上就是一组矩阵乘积。
- **参数 (Parameters) / 系数 (Coefficients) / 权重 (Weights)** : 你要更新的数。
- **损失函数**: 用来告诉你你距离正确结果还有多远的东西。对于分类问题，我们使用交叉熵损失(*cross entropy loss*)，也被称为负对数似然损失(*negative log likelihood loss*)。它惩罚不正确的自信的预测和正确的不自信的预测。

这些模型/识别器/泰迪熊分类器是这样的函数，它们接收像素值、返回概率。开始时有一些 $\vec{y} = X\vec{a}$ 它们使用随机梯度下降拟合参数，来尽量准确地计算你的预测值。现在我们已经学习了怎样做一个变量的回归。后面我们会学习，怎样对多个变量的分类问题做同样的事情，他们基本上是一样的。

这个过程中我们需要用一些数学。我们需要用一些线性代数和微积分。很多人对这个有些担心，告诉我们说：“我不懂数学（not a math person）”。如果你也是这样，没有关系。你错了。你懂数学。事实上在相关的学术研究里，没有“懂数学的人”和“不懂数学的人”。这完全是文化和期望导致的。你需要看看Rachel说的。

[没有不懂数学这回事](#)

If you think you're not good at math...

Proceedings of the National Academy of Sciences of the United States of America

CURRENT ISSUE // ARCHIVE // NEWS & MULTIMEDIA // AUTHORS // ABOUT // COLLECTED ARTICLES // BROWSE BY TOPIC

Check for updates

Female teachers' math anxiety affects girls' math achievement

Sian L. Beilock¹, Elizabeth A. Gunderson, Gerardo Ramirez, and Susan C. Levine

omoju Studies & investigates human and machine intelligence. Advises startups in that space Jan 9 · 5 min read

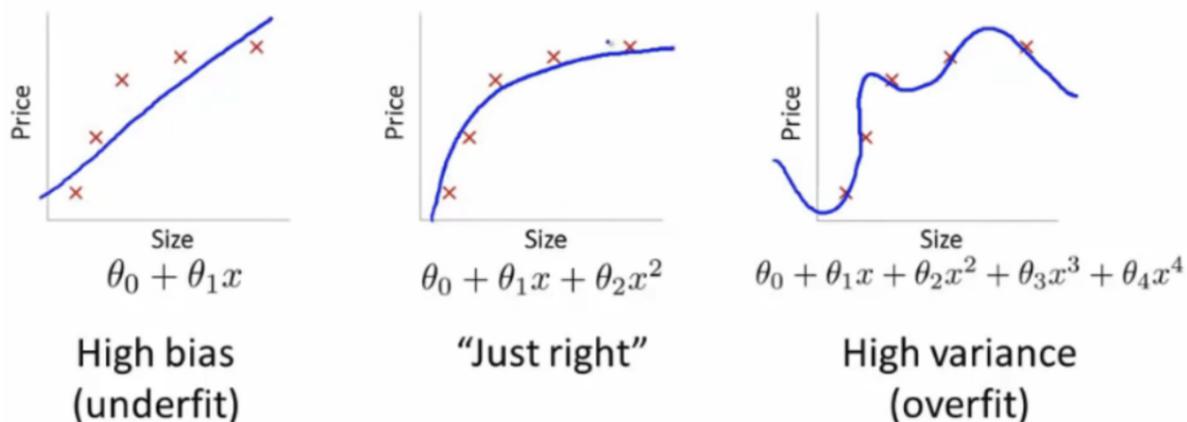
The Myth of Innate Ability in Tech

她会向你介绍一些学术研究。如果你认为你不懂数学，你需要看下这个，嗯，这会让你意识到你是错的，你之所以有这样的想法，是因为有人告诉你你不懂数学。没有学术研究表明有这样一回事。事实上，在罗马和中国的文化里，都没有不懂数学这样一个概念。在一些文化里不会听到，有人说我不懂数学，因为他们的文化都说你特征里没有这回事。

不要被导数，梯度，向量乘积，这些词吓到。你可以学会它们，可以用好它们。

[欠拟合和过拟合 \[1:54:42\]](#)

最后我要说的概念是欠拟合和过拟合。我们刚刚把数据拟合到一条直线上。想象一下，我们的数据根本不是直线形状的。如果我们尝试，把它拟合到一个常数+常数乘以 x （直线），它永远不会拟合得很好。无论你改变这两个系数多少次，它都不会很贴近。



另一方面，我们可能会用更复杂的公式来拟合，这种情况下，它是一个有很多弯曲的高维曲线。如果这样它对其他的气温和冰淇淋销量的数据，不会预测的很好。因为这些曲线太弯曲了。这被叫做过拟合。

我们在寻找能正确地做泰迪熊识别的数学函数。如果你有统计学的背景，你可能会认为，拥有准确数量的参数（没有多余的参数），才能让它正确地识别。这是不正确的。

正则化和验证集 [1:56:07]

有其他的方法能让我们做到不过拟合。通常这被叫做正则化（regularization）。正则化和所有这些方法能让我们的模型不仅仅能在它看到过的数据上工作得很好，也可以在它没有看到的数据上工作得很好。你训练模型时最重要的事情，是知道，它在它没有看到过的数据上工作得怎样。这就是为什么我们需要有验证集，我们下周会学到很多关于这个的内容。

我们使用一个包含泰迪熊、黑熊、灰熊的数据集，做我们的mini-batch随机梯度下降训练循环。完成后，我们用训练时没有使用过的一组图片来检查损失函数和准确度来看看它工作得怎么样。如果我们得到了太弯曲的函数，它会告诉我们，你的损失度和错误率在它没有训练过的熊的图片上表现得很差，因为这些弯曲的部分，是错误的。如果欠拟合，它也会告诉你，你的验证集表现不好。

即使对于没有学习过这个课程，没有学习过深度学习细节的人来说，如果你有管理者或者同事想学习AI，唯一一个你必须教给他们的东西是验证集的概念。因为知道了这个概念，他们就能分辨出别人是不是在忽悠他。他们可以准备一些数据，当别人说：“这是我们要展示的模型”的时候，他们可以说：“很好。我会用这些数据，来检查这个模型，是不是普遍适用的”。当你设计验证集时有很多细节要做好。下周我们会简要讨论这些，但是更完整的版本是Rachel在fasr.ai的blog上发表的名为[怎样（以及为什么要）创建一个好的验证集](#)的文章。这也是，我们在机器学习课程里，要讲很多细节的一个内容。这个课程里，我们会尝试教给你足够的东西。这的确是一个值得深入学习的东西。

谢谢大家！希望你们构建web应用时过得愉快。下周见。