01_intro.ipynb

第一章 深度学习简介

Your Deep Learning Journey

你的深度学习之旅

Hello, and thank you for letting us join you on your deep learning journey, however far along that you may be! In this chapter, we will tell you a little bit more about what to expect in this book, introduce the key concepts behind deep learning, and train our first models on different tasks. It doesn't matter if you don't come from a technical or a mathematical background (though it's okay if you do too!); we wrote this book to make deep learning accessible to as many people as possible.

您好，非常幸运可以参与到您的深度学习学习过程中来，当然也有可能您现在尚在起步阶段！ 在本章节，我们会告诉你一些关于对本书的预期，介绍深度学习背后的关键概念，以及不同的任务中去训练我们的第一个模型。如果你没有技术和数学背景没有关系（当然如果你已经这样做了，那是非常好的！）。通过这本书我们希望可以让尽可能多的人可以可以做深度学习方面的工作。

Deep Learning Is for Everyone

人人可做的深度学习

A lot of people assume that you need all kinds of hard-to-find stuff to get great results with deep learning, but as you'll see in this book, those people are wrong. Here's a few things you absolutely don't need to do world-class deep learning:

很多人认为，你需要所有很难找到的东西才能在机器学习上取得很好的结果，然而在你读完本书后，你会发现那些人是不正确的。相对于世界级深度学习，有些事情你是绝对不需要做的：

Myth (don't need)	Truth
Lots of math	Just high school math is sufficient
Lots of data	We've seen record-breaking results with <50 items of data
Lots of expensive computers	You can get what you need for state of the art work for free

神秘（并不需要）	真相
大量的数学专业知识	只需要高中的数学水平就可以了
海量的数据	我们可以用不到50条数据就可以看到突破行的结果
大量且昂贵的计算机算力	你可以免费获得最先进的工作成果

Deep learning is a computer technique to extract and transform data--with use cases ranging from human speech recognition to animal imagery classification--by using multiple layers of neural networks. Each of these layers takes its inputs from previous layers and progressively refines them. The layers are trained by algorithms that minimize their errors and improve their accuracy. In this way, the network learns to perform a specified task. We will discuss training algorithms in detail in the next section.

深度学习是一门对数据进行抽取与转换的计算机技术，从人类的语音识别到动物的图像分类，通过利用多层神经网络对一系列用例进行处理。每一层神经网络获取其前一层的结果，并对这些输入数据进行提炼处理。通过算法每一层训练以使得的他们的错误最小，并改进他们的处理精度。通过这种方法，网络学会了处理一个特定的任务。我们将在下一节介绍训练算法的细节。

Deep learning has power, flexibility, and simplicity. That's why we believe it should be applied across many disciplines. These include the social and physical sciences, the arts, medicine, finance, scientific research, and many more. To give a personal example, despite having no background in medicine, Jeremy started Enlitic, a company that uses deep learning algorithms to diagnose illness and disease. Within months of starting the company, it was announced that its algorithm could identify malignant tumors more accurately than radiologists.

深度学习具有健壮、柔性和简洁的特性。这也是我们相信它应该被应用于更多领域的原因，这包括社会学、物理、艺术、医学、金融、科学探索等众多学科。以杰里米为例，在没有任何医学背景下，他创建了一家名为Enlitic的公司，他希望公司可以利用深度学习算法去诊断疾病。公司成立几个月内，宣布它们的算法可以比放射科医师更准确的识别恶性肿瘤。

Here's a list of some of the thousands of tasks in different areas at which deep learning, or methods heavily using deep learning, is now the best in the world:

这里列示了不同领域中上千个在任务中的一部分，它们使用了深度学习或大量使用深度学习方法，是这个世界目前为止最好的：

- Natural language processing (NLP): Answering questions; speech recognition; summarizing documents; classifying documents; finding names, dates, etc. in documents; searching for articles mentioning a concept
- 自然语言处理（NLP）：问题回答；语音识别；文件汇总；文档分类；在文档中进行数据搜索等；搜索文章中提到的相关概念
- Computer vision: Satellite and drone imagery interpretation (e.g., for disaster resilience); face recognition; image captioning; reading traffic signs; locating pedestrians and vehicles in autonomous vehicles
- 计算机视觉：卫星与无人机图像理解（例如：防灾能力）；人脸识别；图像字幕；交通标识识别；在自动驾驶中定位行人与车辆
- Medicine:: Finding anomalies in radiology images, including CT, MRI, and X-ray images; counting features in pathology slides; measuring features in ultrasounds; diagnosing diabetic

retinopathy
- 医学：查看各类放射性影像中的异常病理（包括CT，MRI和X光图像）；计算病理幻灯片中的特征；测量超声波特征数；诊断糖尿病视觉病变
- Biology:: Folding proteins; classifying proteins; many genomics tasks, such as tumor-normal sequencing and classifying clinically actionable genetic mutations; cell classification; analyzing protein/protein interactions
- 生物学：寻找蛋白质；蛋白质分类；基因任务处理，例如肿瘤标准测序和临床可诉的基因突变分类；细胞分类；分析蛋白质/蛋白质间的相互影响
- Image generation:: Colorizing images; increasing image resolution; removing noise from images; converting images to art in the style of famous artists
- 图像生成：图像着色；图像分辨率增强；噪影移除；模仿著名艺术家进行图像风格迁移
- Recommendation systems:: Web search; product recommendations; home page layout
- 推荐系统：网页搜索；产品推荐；主页布局
- Playing games:: Chess, Go, most Atari video games, and many real-time strategy games
- 打游戏：国际象棋，围棋，大多雅达利视频游戏和众多实时策略游戏
- Robotics:: Handling objects that are challenging to locate (e.g., transparent, shiny, lacking texture) or hard to pick up
- 机器人：挑战定位处理如透明，耀眼，缺乏质感或非常难以抓取的物体
- Other applications:: Financial and logistical forecasting, text to speech, and much more...
- 其它应用：金融与物流预测，文本转语音，等更多领域...

What is remarkable is that deep learning has such varied application yet nearly all of deep learning is based on a single type of model, the neural network.

深度学习被如此广泛的应用，这是一个非常卓越的成果。然而几乎所有的深度学习都是基于一种模型--神经网络。

But neural networks are not in fact completely new. In order to have a wider perspective on the field, it is worth it to start with a bit of history.

但神经网络并不是一个全新的概念，为了对这个领域有一个更深刻的认识，有必要对神经网络的历史做一个简短的说话。

Neural Networks: A Brief History

神经网络简史

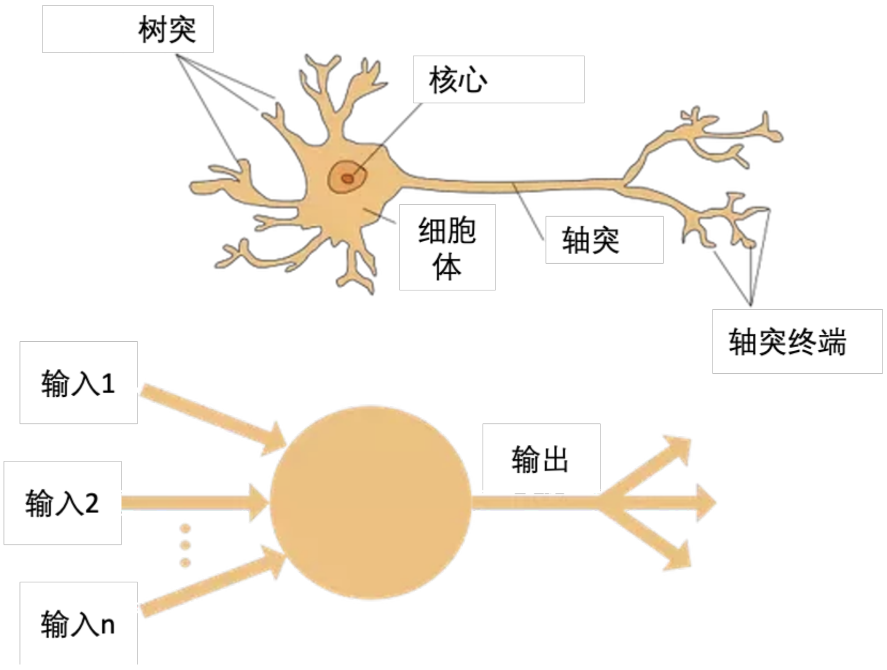
In 1943 Warren McCulloch, a neurophysiologist, and Walter Pitts, a logician, teamed up to develop a mathematical model of an artificial neuron. In their paper "A Logical Calculus of the Ideas Immanent in Nervous Activity" they declared that:

神经生理学专家沃伦·麦卡洛克与逻辑学专家沃尔特·皮茨在1943年合作开发了一个人工神经网络数学模型。在他们的论文“一种在神经活动中固有想法的逻辑计算“中提到：

> :Because of the “all-or-none” character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms.
>
> ：由于神经处理具有“是或非”的活动特征，神经事件以及他们间的相互联系可以用命题逻辑的方法进行处理。发现可以利用这一特性来描述每一个网络的行为。

McCulloch and Pitts realized that a simplified model of a real neuron could be represented using simple addition and thresholding, as shown in . Pitts was self-taught, and by age 12, had received an offer to study at Cambridge University with the great Bertrand Russell. He did not take up this invitation, and indeed throughout his life did not accept any offers of advanced degrees or positions of authority. Most of his famous work was done while he was homeless. Despite his lack of an officially recognized position and increasing social isolation, his work with McCulloch was influential, and was taken up by a psychologist named Frank Rosenblatt.

麦卡洛克和皮茨意识到可以利用简单的加法和阈值来表示一个真实神经元的一个简化模型，如下图<神经元>所示 。皮茨一直都是靠自学，在12岁的时候获得了剑桥大学提供了跟随伯特兰·罗素学习的机会。但他没有接受这个邀请。终生没有接受任何提供给他的高级学位和权威职位机会。他绝大多数著名的研究成果都是在他无家可归的情况下完成的。尽管他缺少权威官方认可的职位和日渐的社会隔离。但他与麦卡洛克的工作成果却有很大的影响力，并被美国著名的心理学家弗兰克·罗森布拉特采用。



图：神经元

Rosenblatt further developed the artificial neuron to give it the ability to learn. Even more importantly, he worked on building the first device that actually used these principles, the Mark I Perceptron. In "The Design of an Intelligent Automaton" Rosenblatt wrote about this work: "We are now about to witness the birth of such a machine–-a machine capable of perceiving, recognizing and identifying its surroundings without any human training or control." The perceptron was built, and was able to successfully recognize simple shapes.

罗森布拉特后面开发了具备学习能力的人工神经元。更重要的是，他得用这一原理构建了第一个设备：马克一号感知器。罗森布拉特在“智能机器人的设计”中对这一工作成果描述到：“我们见证了一个机器的诞生---这台机器在不需要人类对其进行训练与控制的情况下，具备对它的周围感知、辨认和识别能力”。被建造的感知器可以成功的辨认简单的形状。

An MIT professor named Marvin Minsky (who was a grade behind Rosenblatt at the same high school!), along with Seymour Papert, wrote a book called *Perceptrons* (MIT Press), about Rosenblatt's invention. They showed that a single layer of these devices was unable to learn some simple but critical mathematical functions (such as XOR). In the same book, they also showed that using multiple layers of the devices would allow these limitations to be addressed. Unfortunately, only the first of these insights was widely recognized. As a result, the global academic community nearly entirely gave up on neural networks for the next two decades.

麻省理工大学马文·明斯基教授（在同一所高中比罗森布拉特低一个年级）根据罗森布拉特的发明，与西蒙·派珀特合著了《感知器》一书（麻省理工出版）。他们认为一个单层装置不能学到东西，并对数学方程进行了评论（例如XOR）。在同一书中，他们认为这个装置利用多层可以解决这一局限性。不幸的是，只有第一个观点被广泛的承认。导致的结果，之后的20年全球高等院校几乎完全放弃了神经网络。

Perhaps the most pivotal work in neural networks in the last 50 years was the multi-volume *Parallel Distributed Processing* (PDP) by David Rumelhart, James McClellan, and the PDP Research Group, released in 1986 by MIT Press. Chapter 1 lays out a similar hope to that shown by Rosenblatt:

过去50年来，也许神经网络最关键的成果是由大卫·鲁梅尔哈特、詹姆斯·麦克莱伦，以及PDP的研究团队，在1986年通过麻省理工出版社发表的多卷***并行分布式处理***（PDP）。在第一章他们表达了与罗森布拉特相类似的期望。

> :People are smarter than today's computers because the brain employs a basic computational architecture that is more suited to deal with a central aspect of the natural information processing tasks that people are so good at. ...We will introduce a computational framework for modeling cognitive processes that seems... closer than other frameworks to the style of computation as it might be done by the brain.
>
> ：人类比当今的计算机更聪明，因为人脑具有一个基础的计算架构，这个架构更适合处理自然信息处理任务的核心部分，所以在自然信息处理任务这方面人类做的相当好。...我们将要介绍一个计算架构：模式认知处理。相比其它计算架构本架构好像更像是用大脑去处理任务。

The premise that PDP is using here is that traditional computer programs work very differently to brains, and that might be why computer programs had been (at that point) so bad at doing things that brains find easy (such as recognizing objects in pictures). The authors claimed that the PDP approach was "closer than other frameworks" to how the brain works, and therefore it might be better able to handle these kinds of tasks.

前提是，在使用PDP的过程中传统计算机程序工作方式与人脑有巨大差异。这也是为什么在识别图像中的目标时，计算机程序做的如此之差而人脑很容易的原因吧。作者们声称PDP方式相比其它架构更接近人脑的工作方式，因此这种方法可能更好的应对这些类型的工作。

In fact, the approach laid out in PDP is very similar to the approach used in today's neural networks. The book defined parallel distributed processing as requiring:

事实上，在PDP中使用的方式与当今使用的神经网络方式是非常相似的。在书中定义并行分布式处理需要满足如下条件：

1. A set of *processing units*
2. A *state of activation*
3. An *output function* for each unit
4. A *pattern of connectivity* among units
5. A *propagation rule* for propagating patterns of activities through the network of connectivities
6. An *activation rule* for combining the inputs impinging on a unit with the current state of that unit to produce an output for the unit
7. A *learning rule* whereby patterns of connectivity are modified by experience

8. An *environment* within which the system must operate

1. 一组*处理单元*
2. 一个*激活状态*
3. *每个单元有一个输出函数*
4. 单元间的*连接模式*
5. 通过连通性网络传播活动模式的*传播规则*
6. 组合影响一个单元的输入和单元当前状态的*活动规则*，从而生成该单元的输出
7. 一个*学习规则*，从而通过经验修改连通的模式
8. 系统必须运转在其中的一个*环境*

We will see in this book that modern neural networks handle each of these requirements.

我们在这本书中将会看到，神经网络模型涉及到上述所有要求。

In the 1980's most models were built with a second layer of neurons, thus avoiding the problem that had been identified by Minsky and Papert (this was their "pattern of connectivity among units," to use the framework above). And indeed, neural networks were widely used during the '80s and '90s for real, practical projects. However, again a misunderstanding of the theoretical issues held back the field. In theory, adding just one extra layer of neurons was enough to allow any mathematical function to be approximated with these neural networks, but in practice such networks were often too big and too slow to be useful.

20世纪80年代，大多数模型都构造了第二层神经元，因此避免了由明斯基和派珀特发现的问题（上面所提到的框架“单元间连通性模式”）。事实上，在80到90年代期间神经网络已经被广泛使用在真实项目中。然而，对理论问题的误解再次阻碍了这个领域的发展。理论上，只增加一个神经元扩展层就足以允许这些神经网络近似于任何数学函数，但实际上此类网络太大、太慢以至于无法使用。

Although researchers showed 30 years ago that to get practical good performance you need to use even more layers of neurons, it is only in the last decade that this principle has been more widely appreciated and applied. Neural networks are now finally living up to their potential, thanks to the use of more layers, coupled with the capacity to do so due to improvements in computer hardware, increases in data availability, and algorithmic tweaks that allow neural networks to be trained faster and more easily. We now have what Rosenblatt promised: "a machine capable of perceiving, recognizing, and identifying its surroundings without any human training or control."

虽然30年前研究人员已经证明要取提实践上好的表现，你需要使用更多的神经元层，这一原则只是在最近的10几年被广泛的认可与应用。现在神经网络最终发挥出它的潜力，这个要感谢多层的使用，加之计算机硬件的改进，数据可用性的增加，以及算法调整，使得神经网络可以更快与更容易的训练。现在我们实现了罗森布拉特的愿望：“一台机器在不需要人类对其进行训练与控制的情况下，具备对它的周围感知、辨认和识别能力”。

This is what you will learn how to build in this book. But first, since we are going to be spending a lot of time together, let's get to know each other a bit...

这就是你在这本书中将要学习如何去构造的内容。但首先，我们需要一起花费些时间增进对彼此的稍许了解...

Who We Are

我们是谁

We are Sylvain and Jeremy, your guides on this journey. We hope that you will find us well suited for this position.

我们的名字叫西尔维亚和杰里米，是您本次历程的向导。我们希望您能认为我们是称职的。

Jeremy has been using and teaching machine learning for around 30 years. He started using neural networks 25 years ago. During this time, he has led many companies and projects that have machine learning at their core, including founding the first company to focus on deep learning and medicine, Enlitic, and taking on the role of President and Chief Scientist of the world's largest machine learning community, Kaggle. He is the co-founder, along with Dr. Rachel Thomas, of fast.ai, the organization that built the course this book is based on.

杰里米使用并教授机器学习已经有近30年的时间。他在25年前就已经开始使用神经网络。在此期间，他主导了很多公司和项目的机器学习核心工作，包括创建了名为Enlitic的公司，这是第一个专注与深度学习和医疗的公司，并且担任了世界上最大的机器学习社区Kaggle的首席科学家与总裁的职位。他与雷切尔·托马斯博士是fast.ai的联合创始人，这个组织是本书形成的基石。

From time to time you will hear directly from us, in sidebars like this one from Jeremy:

你将一次次的听到来自我们的声音，例如这一次在侧边栏内杰里米的话：

> J: Hi everybody, I'm Jeremy! You might be interested to know that I do not have any formal technical education. I completed a BA, with a major in philosophy, and didn't have great grades. I was much more interested in doing real projects, rather than theoretical studies, so I worked full time at a management consulting firm called McKinsey and Company throughout my university years. If you're somebody who would rather get their hands dirty building stuff than spend years learning abstract concepts, then you will understand where I am coming from! Look out for sidebars from me to find information most suited to people with a less mathematical or formal technical background—that is, people like me…
>
> 杰：嗨..大家好，我是杰里米！你可能会有兴趣知道我并没有任何正式的技术教育。我没有很高的学历只是硕士毕业，主修哲学。我对做真实的项目非常感兴趣，而不是理论研究，所以在我的整个大学学习期间，我的全职工作是在一家名为“麦肯锡”的管理咨询公司。如果你是一个更喜欢动手做一些东西而不想花费太多的时间学习抽象的概念，你可能会理解我的经历。关注侧边栏中我的信息，这更适合那些像我一样缺少数学与正式技术背景的人…

Sylvain, on the other hand, knows a lot about formal technical education. In fact, he has written 10 math textbooks, covering the entire advanced French maths curriculum!

西尔维亚,从别一方面了解一些正式的技术教育。事实上，他已经定了10本学习教科书，覆盖了整个高级法国数学课程！

> S: Unlike Jeremy, I have not spent many years coding and applying machine learning algorithms. Rather, I recently came to the machine learning world, by watching Jeremy's fast.ai course videos. So, if you are somebody who has not opened a terminal and written commands at the command line, then you will understand where I am coming from! Look out for sidebars from me to find information most suited to people with a more mathematical or formal technical background, but less real-world coding experience—that is, people like me…

> 西：不像杰里米，我没有花费太多年的时间在编写代码与应用机器学习算法。而是最近通过观看杰里米的fast.ai课程视频才进入了机器学习世界。所以如果你还没有打开一个终端并在命令行上编写命令，你将会理解我的经历！留意侧边栏内我的信息，这更适合那些像我一样具有丰富的数学知识或正式技术背景，但缺少实际代码编写经验的人...

The fast.ai course has been studied by hundreds of thousands of students, from all walks of life, from all parts of the world. Sylvain stood out as the most impressive student of the course that Jeremy had ever seen, which led to him joining fast.ai, and then becoming the coauthor, along with Jeremy, of the fastai software library.

fast.ai这门课程已经被成千上万个来自世界各地各个年龄段的学生所学习，西尔维亚作是杰出的，杰里米所见过的这个课程最令人印象深刻的学生，从而使得他加入了fast.ai，然后和杰里米成为fastai软件库的联合作者。

All this means that between us you have the best of both worlds: the people who know more about the software than anybody else, because they wrote it; an expert on math, and an expert on coding and machine learning; and also people who understand both what it feels like to be a relative outsider in math, and a relative outsider in coding and machine learning.

所有这些意味着在我们之间你有世界上做好的两个人：这两个人比其它人知道更多的软件，因为这是他们写的；一个数学专业与一个在编写代码和机器学习方面的专业；这两个人也知道不精通数学，以及不熟悉编写代码和机器学习这两类人的感受。

Anybody who has watched sports knows that if you have a two-person commentary team then you also need a third person to do "special comments." Our special commentator is Alexis Gallagher. Alexis has a very diverse background: he has been a researcher in mathematical biology, a screenplay writer, an improv performer, a McKinsey consultant (like Jeremy!), a Swift coder, and a CTO.

任何观看过体育的人都知道，如果你有了两人组的实况报道团队，你也需要一个“特别点评”的第三人，我们的特别评论员是亚历克西斯·加拉格尔。加拉格尔具有丰富多样的背景：他是数学生物研究员，电影编辑，即兴表演家，麦肯锡咨询顾问（像杰里米），敏捷开发者，首席技术官。

> A: I've decided it's time for me to learn about this AI stuff! After all, I've tried pretty much everything else... But I don't really have a background in building machine learning models. Still... how hard can it be? I'm going to be learning throughout this book, just like you are. Look out for my sidebars for learning tips that I found helpful on my journey, and hopefully you will find helpful too.
>
> 亚：我做了个决定“对我来说是时候学习关于这些人工智能的知识了！”毕竟，我已经尝试了很多其它事情... 但我没有实际机器学习模型构建的经验。然而...这到底有多难呢？就像你一样，我计划通过这本书进行学习。请留意我侧边栏，那里有在我学习过程中认为有帮助的学习技巧，希望对你也有所帮助。

How to Learn Deep Learning

如果学习深度学习

Harvard professor David Perkins, who wrote *Making Learning Whole* (Jossey-Bass), has much to say about teaching. The basic idea is to teach the *whole game*. That means that if you're teaching baseball, you first take people to a baseball game or get them to play it. You don't teach them how to wind twine to make a baseball from scratch, the physics of a parabola, or the coefficient of friction of a ball on a bat.

在哈佛教授大卫·珀金斯编写的《使整体学习》一书中已经大量讲述了关于教导的内容。教导的基本思想是*完整参与*。意思是如果你正在教授棒球，你一开始应该是带着他们去一个棒球比赛或让他们玩打棒球。你不可能传授他们从缠绕制作一个棒球开始，物理的抛物线或球体在球棒上的摩擦力系数。

Paul Lockhart, a Columbia math PhD, former Brown professor, and K-12 math teacher, imagines in the influential essay "A Mathematician's Lament" a nightmare world where music and art are taught the way math is taught. Children are not allowed to listen to or play music until they have spent over a decade mastering music notation and theory, spending classes transposing sheet music into a different key. In art class, students study colors and applicators, but aren't allowed to actually paint until college. Sound absurd? This is how math is taught--we require students to spend years doing rote memorization and learning dry, disconnected *fundamentals* that we claim will pay off later, long after most of them quit the subject.

保罗·洛克哈特，哥伦比亚数学博士，前布朗大学教授及K-12数学老师，在他具有影响力的散文"一个数学家的挽歌"中描绘了一个噩梦般的世界，在那里音乐和艺术都以传授数学的方法在教授。孩子们需要花费十年的时间去掌握音乐符号和理论，在此这前孩子们不允许听或演奏音乐，消费阶层将乐谱转换为不同的键。在艺术课堂，学生们学习色彩和涂抹器，但进入大学前都不允许进行实际的绘画。这听起来很荒谬吗？数学就是这样被教授的——我们需要学习花费很多年的时间死记硬背和枯燥的学习，我们声称之后将会有所回报的不连续的基础知识，很久以后绝大多数学生放弃了数学。

Unfortunately, this is where many teaching resources on deep learning begin--**asking learners to follow along with the definition of the Hessian and theorems for the Taylor approximation of your loss functions**, without ever giving examples of actual working code. We're not knocking calculus. We love calculus, and Sylvain has even taught it at the college level, but we don't think it's the best place to start when learning deep learning!

不幸的是，很多入门深度学习教学资源就是这样的：要求学习者遵循海森的定义和定理以对你损失函数进行泰勒逼近，而不提供可以实际运行的代码实例。我们不是在抨击微积分，我们很喜爱它，并且西尔维亚在大学教授微积分，但我们不认为在学习深度学习的时候它是最佳的切入点。

In deep learning, it really helps if you have the motivation to fix your model to get it to do better. That's when you start learning the relevant theory. But you need to have the model in the first place. We teach almost everything through real examples. As we build out those examples, we go deeper and deeper, and we'll show you how to make your projects better and better. This means that you'll be gradually learning all the theoretical foundations you need, in context, in such a way that you'll see why it matters and how it works.

对于深度学习，如果你可以积极的修正你的模型并让模型获得更好的结果，这是非常有帮助的。即使你开始学习相关原理，但也你需要把模型放在首要的位置。我们传授的每个内容几乎都有实际的例子。通过构建的这些实例我们逐步深入，并用我们会给你展示如果让你的项目结果越来越好。这意味着你会逐步学习到所有需要的基础原理，通过上下文内容，并用这种方法你将会明白它什么是这个样子和他们如何工作的。

So, here's our commitment to you. Throughout this book, we will follow these principles:

所以在这里我们承诺，贯穿这本书始终，我们会遵循如下原则：

- Teaching the *whole game*. We'll start by showing how to use a complete, working, very usable, state-of-the-art deep learning network to solve real-world problems, using simple, expressive tools. And then we'll gradually dig deeper and deeper into understanding how those tools are made, and how the tools that make those tools are made, and so on…
- 完整游戏教学法。我们将从展示如何使用一个完整、可运行、完全可用、时下最流行的深度学习网络开始去解决现实世界的问题，使用简单，描述生动的工具。然后我们会逐步加深并深入理解那些工作是如何制作的，以及制作那些工具的工具是如何制作的，等等…
- Always teaching through examples. We'll ensure that there is a context and a purpose that you can understand intuitively, rather than starting with algebraic symbol manipulation.
- 总是实例教学。我们保证你可以凭直觉就可以理解其内容与用途，而不是从操作代数符号开始。
- Simplifying as much as possible. We've spent years building tools and teaching methods that make previously complex topics very simple.
- 尽可能的简单明了。我们已经花费多年的时间构建这些工具和教学方法，以确保之前前面那些复杂的主题变的非常简单。
- Removing barriers. Deep learning has, until now, been a very exclusive game. We're breaking it open, and ensuring that everyone can play.
- 移除障碍。直到现在，深度学习依然是一个非常高档的游戏。我们正在打开它，并确保每一个人都可以玩这个游戏。

The hardest part of deep learning is artisanal: how do you know if you've got enough data, whether it is in the right format, if your model is training properly, and, if it's not, what you should do about it? That is why we believe in learning by doing. As with basic data science skills, with deep learning you only get better through practical experience. Trying to spend too much time on the theory can be counterproductive. The key is to just code and try to solve problems: the theory can come later, when you have context and motivation.

深度学习最难的部分是动手部分：你如何知道是否已经有了足够的数据，它的格式是否正确，你的模型是否正确的训练，并且如果不是，你应该对它做什么？这主是为什么我们相信学习过程中要手工操作。基于基本的数据科学技能，对于深度学习你只有通过实际经验才可以做的更好。尝试花费太多的时间在理论上面只会适得其反。重点是通过代码和尝试解决问题：当你在具有了一定的环境和动机，理论会随之而来。

There will be times when the journey will feel hard. Times where you feel stuck. Don't give up! Rewind through the book to find the last bit where you definitely weren't stuck, and then read slowly through from there to find the first thing that isn't clear. Then try some code experiments yourself, and Google around for more tutorials on whatever the issue you're stuck with is—often you'll find some different angle on the material might help it to click. Also, it's expected and normal to not understand everything (especially the code) on first reading. Trying to understand the material serially before proceeding can sometimes be hard. Sometimes things click into place after you get more context from parts down the road, from having a bigger picture. So if you do get stuck on a section, try moving on anyway and make a note to come back to it later.

在学习过程中将会多次感到遇到困难，多次感觉被问题困住，千万不要放弃！找回到最近明确没有困难的位置，并且从那个地方开始慢慢阅读，去思考最开始不明白的东西是什么。然后尝试自己做一下代码实验，并且可以去谷歌或百度找更多与你卡住相关的问题辅导资料，通常你会发现有多种不同的解决方案。当然，可以预见到且正常的是，在最初读这本书时不可能对什么事情都理解（尤其是代码）。在行动这前可能有时会有困难，尝试理解按顺序理解这些内容。有时候在你从其他地方、从一个更宏观的整

体获取到更多的背景信息，问题往往会很自然的被解决。所以如果你在某个环节遇到问题，尝试尽可能的继续并做笔记，然后晚些时候再回来看这一部分。

Remember, you don't need any particular academic background to succeed at deep learning. Many important breakthroughs are made in research and industry by folks without a PhD, such as "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks"—one of the most influential papers of the last decade—with over 5,000 citations, which was written by Alec Radford when he was an undergraduate. Even at Tesla, where they're trying to solve the extremely tough challenge of making a self-driving car, CEO Elon Musk says:

记住，要想在机器学习上取的成功，你不需要任何特定的大学背景。许多在研究与工业领域取得的重大突破都是由那些没有博士学位的人取得的，例如：“基于深度卷积生成对抗网络的无监督表示学习”，在过去的10年这篇文章是最有影响力的论文之一，有超过5000次的引用，写这篇论文的作者亚历克·雷德福当时仅仅是一名大学生。即使在特思拉，他们尝试了一个终极挑战：制造一辆自动驾驶的汽车。首席执行管伊隆·马斯克说到：

> : A PhD is definitely not required. All that matters is a deep understanding of AI & ability to implement NNs in a way that is actually useful (latter point is what’s truly hard). Don’t care if you even graduated high school.
>
> 伊：明确的说博士学位是不需要的。在某种程度上，所有的事情是对人工智能的深度理解，以及具备实施神经网络的能力，这是非常有用的（后者真的很难）。如果你只有本科学历所以请不要在意。

What you will need to do to succeed however is to apply what you learn in this book to a personal project, and always persevere.

如果你希望在深度学习上取得成功，你需要做的事情是：把你在这本书里所学到的东西应用到个人的项目中去，并坚持不懈！

Your Projects and Your Mindset

你的项目和你的心态

Whether you're excited to identify if plants are diseased from pictures of their leaves, auto-generate knitting patterns, diagnose TB from X-rays, or determine when a raccoon is using your cat door, we will get you using deep learning on your own problems (via pre-trained models from others) as quickly as possible, and then will progressively drill into more details. You'll learn how to use deep learning to solve your own problems at state-of-the-art accuracy within the first 30 minutes of the next chapter! (And feel free to skip straight there now if you're dying to get coding right away.) There is a pernicious myth out there that you need to have computing resources and datasets the size of those at Google to be able to do deep learning, but it's not true.

你是否兴奋与如果植物有了疾病，通过叶子图片就可以识别出来，自动生成编制图案，通过X光片诊断结核病，或决策什么时候浣熊可以使用你的喵星人（猫）大门。通过从其它地方获取到的预训练模型，我们将尽可能快的帮助你将深度学习用于解决你的个人问题，然后逐渐深入更多细节。在下一章开始的30分钟，你将学习如何利用深度学习以最先进的精度去解决你的个人问题！（如果你迫切的想立刻写代码，你现在可以直接到那一部分）存在这么一种有害的传说：你需要具有类似谷歌公司这样的计算资源和数据集才可以做深度学习，然而事实并不是这样的。

So, what sorts of tasks make for good test cases? You could train your model to distinguish between Picasso and Monet paintings or to pick out pictures of your daughter instead of pictures of your son. It helps to focus on your hobbies and passions–-setting yourself four or five little projects rather than striving to solve a big, grand problem tends to work better when you're getting started. Since it is easy to get stuck, trying to be too ambitious too early can often backfire. Then, once you've got the basics mastered, aim to complete something you're really proud of!

那么，那些类型的任务适合做测试工作？你可以训练你的模型把毕加索和莫奈的画作区分出来或辨认出你女儿的图片而不是你的儿子。设置4、5个你自己的小项目而不是一个非常艰难的大项目，这会帮助你聚焦在你的爱好上并会富有激情。在你起步阶段，重要的问题会让工作做的更好。尝试做一些过于富有野心容易事得其反，因为很容易被难住。然后，一旦你打牢基础，就可以去完成那些你真正引以为豪的与了。

> J: Deep learning can be set to work on almost any problem. For instance, my first startup was a company called FastMail, which provided enhanced email services when it launched in 1999 (and still does to this day). In 2002 I set it up to use a primitive form of deep learning, single-layer neural networks, to help categorize emails and stop customers from receiving spam.
>
> 杰：深度学习能够被应用于几乎所有问题。例如，我最开始创建了一家名为FastMail的公司，在创立之初这家公司提供邮件增值服务（直到现在也在做这个事）。在2002年我为它构建并使用了一个深度学习的简化版（只有一层神经网络），以帮助邮件进行分类和帮客户过滤垃圾邮件。

Common character traits in the people that do well at deep learning include playfulness and curiosity. The late physicist Richard Feynman is an example of someone who we'd expect to be great at deep learning: his development of an understanding of the movement of subatomic particles came from his amusement at how plates wobble when they spin in the air.

在深度学习方面做的好的人的共同特征包括兴趣与好奇。对于希望在深度学习方面做的更好的人来说，已故物理学家理查德·费曼就是一个很好的例子：他对亚粒子颗粒运动的理解的发展就源自对板在空中旋转里是如何摇摆的兴趣。

Let's now focus on what you will learn, starting with the software.

现在让我们聚焦于将要学习的内容，先从软件开始。

The Software: PyTorch, fastai, and Jupyter

软件：PyTorch，fastai 和 Jupyter

(And Why It Doesn't Matter)

（并且为什么它不重要）

We've completed hundreds of machine learning projects using dozens of different packages, and many different programming languages. At fast.ai, we have written courses using most of the main deep learning and machine learning packages used today. After PyTorch came out in 2017 we spent over a thousand hours testing it before deciding that we would use it for future courses, software development, and research. Since that time PyTorch has become the world's fastest-growing deep learning library and is already used for most research papers at top conferences. This is generally a leading indicator of usage in industry, because these are the papers that end up getting used in products and services commercially. We have found that PyTorch is the most flexible and expressive library for deep learning. It does not trade off speed for simplicity, but provides both.

我们已经利用数打不同的包和很多不同的程序语言完成了上百个机器学习的项目。在Fast.AI，我们编写的课程使用了当今主流深度学习和机器学习程序包的大多数。在2017年PyTorch面世后，在决定是否把它应用到我们未来的课程、软件开发和研究中之前，我们花费了超过上千个小时对它进行测试。从那时起，PyTorch已经成为世界上成长速度最快的深度学习库，并且已经被应用于绝大多数顶会的研究论文中。通常这是工业界使用的风向标，因为这些论文最终是要被商业生产和服务采用的。我们已经发现对于深度学习PyTorch是最灵活和表现力的库。它没有为了简洁把速度放弃，而同时具备了这两个特性。

PyTorch works best as a low-level foundation library, providing the basic operations for higher-level functionality. The fastai library is the most popular library for adding this higher-level functionality on top of PyTorch. It's also particularly well suited to the purposes of this book, because it is unique in providing a deeply layered software architecture (there's even a peer-reviewed academic paper about this layered API). In this book, as we go deeper and deeper into the foundations of deep learning, we will also go deeper and deeper into the layers of fastai. This book covers version 2 of the fastai library, which is a from-scratch rewrite providing many unique features.

作为一个底层基础库PyTorch工作的非常好，它也提供了对于上层功能设计的基础操作。fastai库作为基于PyTorch的上层功能设计层是最流行的库。它也非常契合这本书的用途，因为它在提供一个深度层软件架构方面是唯一无二的（这里甚至有一个关于本层API的同行评审学术论文）。在这本书中，我会逐步深入到深度学习的基础（或底层）部分，我们也将会逐步深入到fastai的各个层。这本书涵盖了fastai版本2.0的库，这都是从头重编写提供了许多唯一无二的特性。

However, it doesn't really matter what software you learn, because it takes only a few days to learn to switch from one library to another. What really matters is learning the deep learning foundations and techniques properly. Our focus will be on using code that as clearly as possibly expresses the concepts that you need to learn. Where we are teaching high-level concepts, we will use high-level fastai code. Where we are teaching low-level concepts, we will use low-level PyTorch, or even pure Python code.

然而，你所学习的软件内容并不是非常重要，因为只需要花费几天的学习成本就可以从一个库切换到其它库。真正重要的是学习正确的深度学习基础和技术。我们将聚焦在尽可能使用简洁的代码来表述你需要学习的概念。在我们传授高级概念的地方，会将使用上层fastai代码。在我们传授基础概念的时候，会用到PyTorch级，甚至会用纯Python代码。

If it feels like new deep learning libraries are appearing at a rapid pace nowadays, then you need to be prepared for a much faster rate of change in the coming months and years. As more people enter the field, they will bring more skills and ideas, and try more things. You should assume that whatever specific libraries and software you learn today will be obsolete in a year or two. Just think about the number of changes in libraries and technology stacks that occur all the time in the world of web programming—a much more mature and slow-growing area than deep learning. We strongly believe that the focus in learning needs to be on understanding the underlying techniques and how to apply them in practice, and how to quickly build expertise in new tools and techniques as they are released.

现如今给人的感觉好像新的深度学习库以很快的速度出现，你需要做的准备是在今后的年月里高效的去切换。做为一个很多人参与的领域，他们将会带来更多的技术与想法，并尝试做（或搞）更多的事情。你应该做好假设，今天你无论学到多么特殊的库和软件在一到二年的时间都会被淘汰。只需要想一直以来在网络编程的世界里软件库和技术栈变化的次数，这是一个相比深度学习更成熟和低速成长的领域。我们坚信学习的重心应该放在理解基础技术和如何把他们应用到实践中去，并且新的工具和技术发布后如何快速构建专长。

By the end of the book, you'll understand nearly all the code that's inside fastai (and much of PyTorch too), because in each chapter we'll be digging a level deeper to show you exactly what's going on as we build and train our models. This means that you'll have learned the most important best practices used in modern deep learning—not just how to use them, but how they really work and are implemented. If you want to use those approaches in another framework, you'll have the knowledge you need to do so if needed.

在书的结尾部分，你将会理解几乎所有fastai（还有很多PyToch）的内部代码，因为在每一章我们都会进入一个更深的等级，去展示那些作为我们构建和训练的模型是怎么运行的。这意味着你将会学到用在当今深度学习最重要的最佳实践：不仅仅是如何使用他们，而是他们实际怎么工作和实现的。如果你想在其它架构使用那些方法，你将具备了你要做这些事的知识（如果需要的话）。

Since the most important thing for learning deep learning is writing code and experimenting, it's important that you have a great platform for experimenting with code. The most popular programming experimentation platform is called Jupyter. This is what we will be using throughout this book. We will show you how you can use Jupyter to train and experiment with models and introspect every stage of the data pre-processing and model development pipeline. Jupyter Notebook is the most popular tool for doing data science in Python, for good reason. It is powerful, flexible, and easy to use. We think you will love it!

所以学习深度学习最重要的事情是写代码和做实验，你有一个很棒的平台用于做代码实验是很重要的。最受欢迎的编码实验平台被称为Jupyter，贯穿本书我们都会用到它。我们会展示，你能用Jupyter怎么去训练和实验模型和窥探预处理和模型开发流水线的每一步。对于用Python做数据科学Jupyter Notebook是最爱欢迎的工具，这是有充分的理由的，它是一个强大、灵活和容易使用的工具。我们想你将会喜爱它的。

Let's see it in practice and train our first model.

让我们看看它实践和训练我们的第一个模型。

Your First Model

你的第一个模型

As we said before, we will teach you how to do things before we explain why they work. Following this top-down approach, we will begin by actually training an image classifier to recognize dogs and cats with almost 100% accuracy. To train this model and run our experiments, you will need to do some initial setup. Don't worry, it's not as hard as it looks.

正如我们之前所说，在我们解释为什么他们这样之前我们先教你如何去做事。遵循这事从顶向下的方法，我们将通过实际训练一个图像分类器开始，去以近乎100%准确率识别狗和猫。去训练这个模型并运行我们的实验，你需要去做一些初始设置。不要担心，它没有看起来那么难。

> s: Do not skip the setup part even if it looks intimidating at first, especially if you have little or no experience using things like a terminal or the command line. Most of that is actually not necessary and you will find that the easiest servers can be set up with just your usual web browser. It is crucial that you run your own experiments in parallel with this book in order to learn.
>
> 西：即使在一开始它看起来很唬人，尤其如果你只有很少或没有使用类似命令终端或行命令经验的情况下，请务必不要略过这部分。他们大多数内部实际上不是必须的，并且你会发现最简单的那此服务只需用普通的网络浏览器就可以进行设置。用这本书进行学习的同时运行你的实验是至关重要的。

Getting a GPU Deep Learning Server

获取一台GPU深度学习服务器

To do nearly everything in this book, you'll need access to a computer with an NVIDIA GPU (unfortunately other brands of GPU are not fully supported by the main deep learning libraries). However, we don't recommend you buy one; in fact, even if you already have one, we don't suggest you use it just yet! Setting up a computer takes time and energy, and you want all your energy to focus on deep learning right now. Therefore, we instead suggest you rent access to a computer that already has everything you need preinstalled and ready to go. Costs can be as little as US$0.25 per hour while you're using it, and some options are even free.

在这本书做几乎所有的事情，你都需要拥有一台带有英伟达GPU卡的计算机，因为很不幸的事其它品牌的GPU不能完全支持主要的深度学习库。然而，我们并不推荐你去买一台。事实上，即使你已经有一台，我们也不建议你用它！配置一台计算机需要花费时间和精力，而此时此刻的你想要把所有的精力聚焦在深度学习。因此，我们的替代方案是，你通过租赁获取一台已经预先安装你需要的所有东西并可以随时使用的机器。在使用它时你每小时仅只需要花费25美分，并且一些操作甚至是免费的。

> jargon: Graphics Processing Unit (GPU): Also known as a *graphics card*. A special kind of processor in your computer that can handle thousands of single tasks at the same time, especially designed for displaying 3D environments on a computer for playing games. These same basic tasks are very similar to what neural networks do, such that GPUs can run neural networks hundreds of times faster than regular CPUs. All modern computers contain a GPU, but few contain the right kind of GPU necessary for deep learning.
>
> 行话（黑话）：图形处理器（GPU）：又称为图形卡。在你计算机里的一种特殊处理器，它能同时处理数千个任务，这是为玩的游戏在计算机中显示3D环境而特别设计的。此类基础任务与神经网络做的事情是非常相似的，因此GPU运行神经网络能够比常规CPU快数百倍。所有现代计算机都包含一块GPU，但很少包含深度学习所必须的那种GPU。

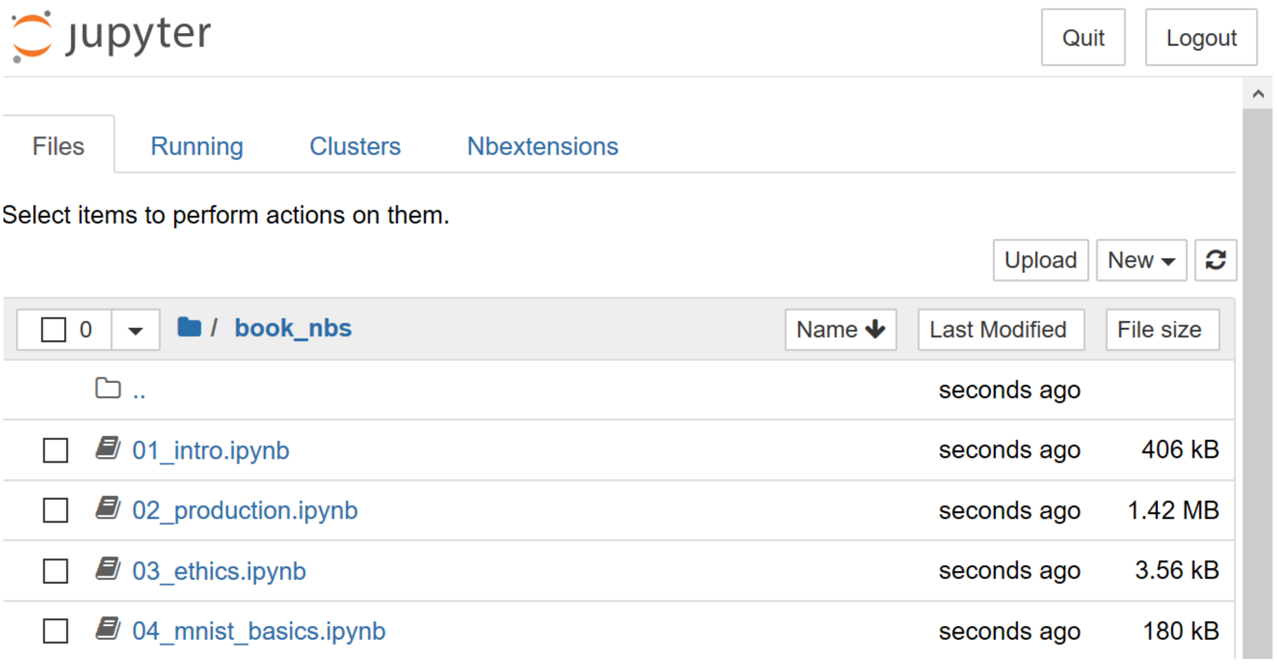
The best choice of GPU servers to use with this book will change over time, as companies come and go and prices change. We maintain a list of our recommended options on the book's website, so go there now and follow the instructions to get connected to a GPU deep learning server. Don't worry, it only takes about two minutes to get set up on most platforms, and many don't even require any payment, or even a credit card, to get started.

随着公司的来来去去和价格变化，本书所使用GPU的最佳选择会随着时间发生变化。在本书网站我们维护了一个建议列表，所以现在就到那里，并且跟随建议去连接一个GPU深度学习服务器。不要担心，在大多数平台配置工作只需要差不多2分钟，并且很多甚至不需要支付任何费用，或者不需要信用卡就可以开始。

> A: My two cents: heed this advice! If you like computers you will be tempted to set up your own box. Beware! It is feasible but surprisingly involved and distracting. There is a good reason this book is not titled, *Everything You Ever Wanted to Know About Ubuntu System Administration, NVIDIA Driver Installation, apt-get, conda, pip, and Jupyter Notebook Configuration*. That would be a book of its own. Having designed and deployed our production machine learning infrastructure at work, I can testify it has its satisfactions, but it is as unrelated to modeling as maintaining an airplane is to flying one.
>
> 亚：我的两美分：听从这个建议！如果你喜爱计算机你可能会迷恋配置自己的工作平台。请注意！虽然这是可行的，但会被过度的牵扯和分散精力。基于这个原因这本书没有标题。*你希望知道所有的事情：乌班图系统管理、英伟达驱动安装、apt-get、conda和Jupyter Notebook的配置*。它们每一项也许都可以是一本书。设计和布置我们生产机器学习工作的基础设施，我能够证明它是满足的，但它与模型无关，就像维护一架飞行就是在驾驶一架飞机一样。

Each option shown on the website includes a tutorial; after completing the tutorial, you will end up with a screen looking like <notebook_init>.

在网站上每一个操作都包含了一个指引，完成所有指引后你最终在屏幕上会看到类似下图<笔记初始>。



Select items to perform actions on them.

```
/ book_nbs
..                        seconds ago
01_intro.ipynb            seconds ago   406 kB
02_production.ipynb       seconds ago   1.42 MB
03_ethics.ipynb           seconds ago   3.56 kB
04_mnist_basics.ipynb     seconds ago   180 kB
```

图：Jupyter Notebook初始页面

You are now ready to run your first Jupyter notebook!

你现在准备好运行你的第一个Jupyter notebook了！

> jargon: Jupyter Notebook: A piece of software that allows you to include formatted text, code, images, videos, and much more, all within a single interactive document. Jupyter received the highest honor for software, the ACM Software System Award, thanks to its wide use and enormous impact in many academic fields and in industry. Jupyter Notebook is the

> software most widely used by data scientists for developing and interacting with deep learning models.
>
> 行话（黑话）：Jupyter Notebook: 这是一款允许你包含格式文本、代码、图片、视频和更多内容的软件，所以内容都在单一的交互文档。Jupyter收到了软件方面的最高冗余：ACM软件系统奖。用于表彰在众多学术领域和工业领域广泛的使用和巨大的影响力。Jupyter Notebook是一款最广泛的被数据科学家用于深度模型开发和交互的软件。

Running Your First Notebook

运行你的第一个Notebook

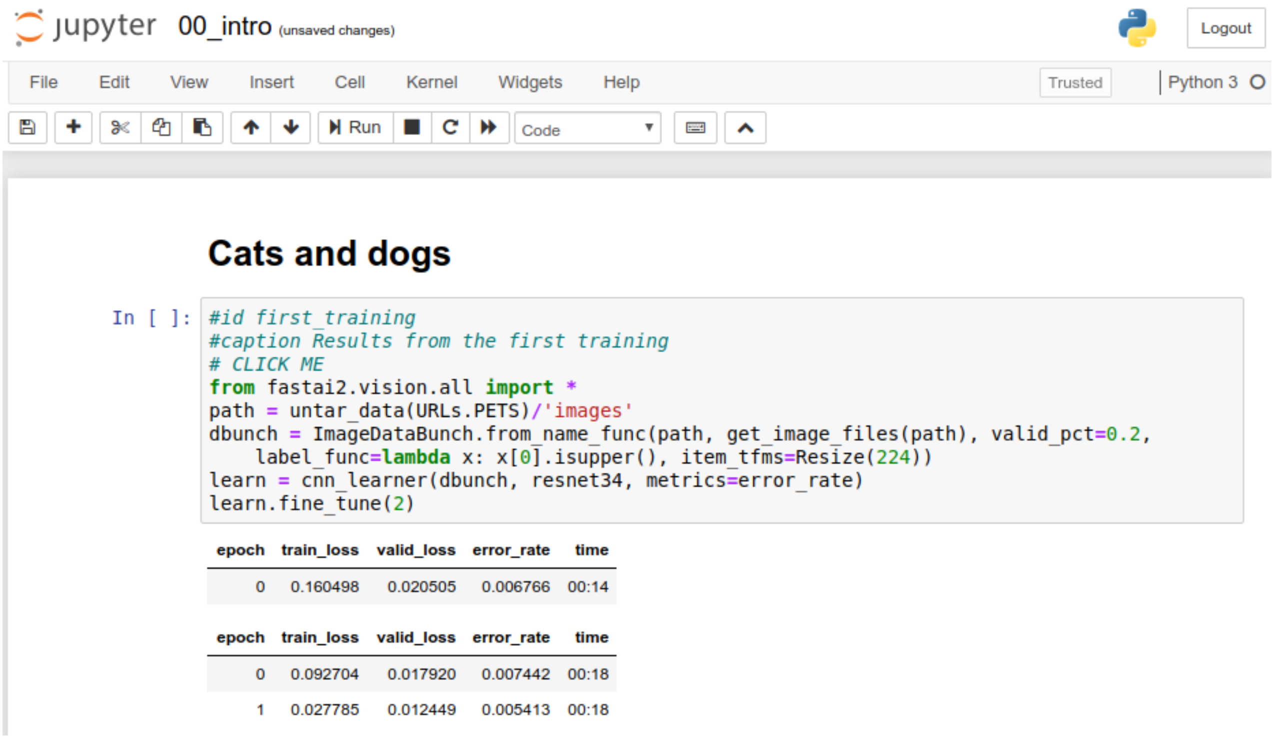
The notebooks are labeled by chapter and then by notebook number, so that they are in the same order as they are presented in this book. So, the very first notebook you will see listed is the notebook that you need to use now. You will be using this notebook to train a model that can recognize dog and cat photos. To do this, you'll be downloading a *dataset* of dog and cat photos, and using that to *train a model*. A dataset is simply a bunch of data—it could be images, emails, financial indicators, sounds, or anything else. There are many datasets made freely available that are suitable for training models. Many of these datasets are created by academics to help advance research, many are made available for competitions (there are competitions where data scientists can compete to see who has the most accurate model!), and some are by-products of other processes (such as financial filings).

这些笔记已经通过章节进行了标记，然后对笔记本做了编号，这样在本书中他们所呈现的次序就相同了。所以在最顶端你看到的笔记就是你现在需要用的到这个。你将用这个笔记去训练一个能够识别狗猫图片的模型。要做这个事情，你需要下载一个狗猫图片数据集，并用这个数据集训练模型。一个数据集是一个简单的数据分支，它可以是图片、电子邮件、财务指标、声音或任何其它东西。这里有许多免费可以适合做模型训练的有效数据集。其中一些数据集的创建是为了学术用于高端研究，有些数据集被制作出来的目的是为了比赛（数据科学家通过比赛可以对比出谁的模型准确率是最高的），还有一些只是其它工作处理的副产品（例如金融数据存档）。

> note: Full and Stripped Notebooks: There are two folders containing different versions of the notebooks. The *full* folder contains the exact notebooks used to create the book you're reading now, with all the prose and outputs. The *stripped* version has the same headings and code cells, but all outputs and prose have been removed. After reading a section of the book, we recommend working through the stripped notebooks, with the book closed, and seeing if you can figure out what each cell will show before you execute it. Also try to recall what the code is demonstrating.
>
> 注意：全版和简版笔记：这里有两个包含了不同版本笔记的目录。全版目录包含了正文和输出全部内容，用于你现在正在阅读的这本书。简版具有与全版相同的标题和代码单元，只是所有的输出和正文都被移除了。读完这本书的一个章节后，我们建议合上这本书，打开对应的简版笔记，看你在执行每一个代码单元前是否可以想像出他们的输出结果，还要尝试回想代码操作示范的内容。

To open a notebook, just click on it. The notebook will open, and it will look something like (note that there may be slight differences in details across different platforms; you can ignore those differences).

要打开一个笔记只需要用鼠标点击它即可。笔记将会打开并且它看起来像（不同的平台笔记展示可能做有少许细小差别，你可以忽略这些差异）。


jupyter 00_intro (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

Cats and dogs

```
In [ ]: #id first_training
        #caption Results from the first training
        # CLICK ME
        from fastai2.vision.all import *
        path = untar_data(URLs.PETS)/'images'
        dbunch = ImageDataBunch.from_name_func(path, get_image_files(path), valid_pct=0.2,
            label_func=lambda x: x[0].isupper(), item_tfms=Resize(224))
        learn = cnn_learner(dbunch, resnet34, metrics=error_rate)
        learn.fine_tune(2)
```

```
epoch  train_loss  valid_loss  error_rate  time
    0    0.160498    0.020505    0.006766  00:14

epoch  train_loss  valid_loss  error_rate  time
    0    0.092704    0.017920    0.007442  00:18
    1    0.027785    0.012449    0.005413  00:18
```

图：一个Jupyter Notebook显示页面

A notebook consists of *cells*. There are two main types of cell:

一个笔记本的单元构成主要有两种类型：

- Cells containing formatted text, images, and so forth. These use a format called *markdown*, which you will learn about soon.
- 一类单元包含格式：正式文本、图像，以及常用的markdown文本编辑格式，这类编辑格式后面我们会很快学到。
- Cells containing code that can be executed, and outputs will appear immediately underneath (which could be plain text, tables, images, animations, sounds, or even interactive applications).
- 另一类单元包含可以执行的代码，并能在单元下面立即显示输出结果（输出的内容可以是纯文本、表格、图片、动画、声音，甚至交互应用）

Jupyter notebooks can be in one of two modes: edit mode or command mode. In edit mode typing on your keyboard enters the letters into the cell in the usual way. However, in command mode, you will not see any flashing cursor, and the keys on your keyboard will each have a special function.

Jupyter notebook 可以处于以下两种模式之一：编辑模式或命令模式 。在编辑模式下以常用的方式用键盘在笔记单元中录入文本。然而，在命令模式下，你不会看到任何光标，在你拍键盘上相关的键都有一个特定的功能。

Before continuing, press the Escape key on your keyboard to switch to command mode (if you are already in command mode, this does nothing, so press it now just in case). To see a complete list of all of the functions available, press H; press Escape to remove this help screen. Notice that in command mode, unlike most programs, commands do not require you to hold down Control, Alt, or similar—you simply press the required letter key.

继续之前，按你键盘上的ESC键切换到命令模式（如果你已经处于命令模式下不会有任何变化。防万一，现在按一下ESC键以确保处于命令模式），按H键可以看到所有有效功能的完整列表；按ESC按退出帮助页面。请注意，不像其它大多数程序，在命令模式下，不需要你保持按住Control，Alt或其它类似的功能键去控制，你只需要简单的按相关字母键即可。

You can make a copy of a cell by pressing C (the cell needs to be selected first, indicated with an outline around it; if it is not already selected, click on it once). Then press V to paste a copy of it.

你通过按字母 C 键能拷贝一个单元内容（需要拷贝的笔记单元要首先被选中，有一个指示轮廓线围绕这个单元格，如果没有只需要点击一下它），然后按字母V键粘贴这个拷贝。

Click on the cell that begins with the line "# CLICK ME" to select it. The first character in that line indicates that what follows is a comment in Python, so it is ignored when executing the cell. The rest of the cell is, believe it or not, a complete system for creating and training a state-of-the-art model for recognizing cats versus dogs. So, let's train it now! To do so, just press Shift-Enter on your keyboard, or press the Play button on the toolbar. Then wait a few minutes while the following things happen:

点击笔记单元去选中它，里面的内容第一行是“#CLICK ME”。这一行的首字母#指示这一行的内容是Python的注释，所以在执行本单元格代码时它会被忽略。不管你信不信，单元格的剩余内容是一个完整系统，为了创建和训练一个最先进的狗猫识别模型。所以，让我们开始训练模式吧！只用按你键盘上的Shift+Enter组合键或按单元工具条上的运行按钮就可以开始了。然后等上几分钟接下来会发生下面的事情：

1. A dataset called the Oxford-IIIT Pet Dataset that contains 7,349 images of cats and dogs from 37 different breeds will be downloaded from the fast.ai datasets collection to the GPU server you are using, and will then be extracted.
2. A *pretrained model* that has already been trained on 1.3 million images, using a competition-winning model will be downloaded from the internet.
3. The pretrained model will be *fine-tuned* using the latest advances in transfer learning, to create a model that is specially customized for recognizing dogs and cats.

1. 将会从fast.ai数据集仓库下载一个被称为生津IIIT宠物数据集，里面包含了37种不同品种的7349张狗和猫的图片收集到你正在使用的GPU服务器，然后抽取 图片。
2. 利用一百三十万张图片训练出的一个*预训练模型* 将会从因特网上被下载，这是一个用于赢得比赛的模式。
3. 使用最进展的迁移学习对这个预训练模型进行微调*，去生成一个为识别狗和猫而特制的模型。

The first two steps only need to be run once on your GPU server. If you run the cell again, it will use the dataset and model that have already been downloaded, rather than downloading them again. Let's take a look at the contents of the cell, and the results (<>):

头两步只会在你的GPU服务器上运行一次。如果你再次运行这个单元格，它将会使用已经下载过的数据集和模型，而还是再次下载一遍。让我们看一下单元格里内容和运行结果。

```
# CLICK ME
#id first_training
#caption Results from the first training
from fastai.vision.all import *
path = untar_data(URLs.PETS)/'images'

def is_cat(x): return x[0].isupper()
dls = ImageDataLoaders.from_name_func(
    path, get_image_files(path), valid_pct=0.2, seed=42,
    label_func=is_cat, item_tfms=Resize(224))

learn = cnn_learner(dls, resnet34, metrics=error_rate)
learn.fine_tune(1)
```

epoch（轮次）	train_loss（训练损失）	valid_loss（验证损失）	error_rate（错误率）	time（时间）
0	0.169390	0.021388	0.005413	00:14

epoch（轮次）	train_loss（训练损失）	valid_loss（验证损失）	error_rate（错误率）	time（时间）
0	0.058748	0.009240	0.002706	00:19

You will probably not see exactly the same results that are in the book. There are a lot of sources of small random variation involved in training models. We generally see an error rate of well less than 0.02 in this example, however.

在这本书你可能不会看到同样精度的结果。训练模型存在相当多小的随机变化。然而在这个例子中我们通常看小于0.02的错误率。

> important: Trianing Time: Depending on your network speed, it might take a few minutes to download the pretrained model and dataset. Running `fine_tune` might take a minute or so. Often models in this book take a few minutes to train, as will your own models, so it's a good idea to come up with good techniques to make the most of this time. For instance, keep reading the next section while your model trains, or open up another notebook and use it for some coding experiments.
>
> 重要提示：训练时间：根据你的网速，它可能会花些时间下载预训练模型和数据集。运行`微调`可能会花1分钟或更多。在本书中做为你将拥有的模型通常会花费几分钟的时间去训练，所以想出好的方法充分利用这段时间是一个很好的想法。例如，在你训练模型的同时你可以阅读后面的章节，或打开别一个笔记本并用它做一些代码实验。

Sidebar: This Book Was Written in Jupyter Notebooks

侧边栏：这本书是在 Jupyter Notebooks 中编写

We wrote this book using Jupyter notebooks, so for nearly every chart, table, and calculation in this book, we'll be showing you the exact code required to replicate it yourself. That's why very often in this book, you will see some code immediately followed by a table, a picture or just some text. If you go on the book's website you will find all the code, and you can try running and modifying every example yourself.

我们使用 Jupyter notebooks 工具编写的这本书，所以在书中几乎每一个图、表和计算我们都会给你呈现你自己可需复制的精确代码。这就是为什么你会经常看到在一个表、图或相关文本后面立刻会跟随一些代码。如果你前往本书的网站你会看到所有代码，你能够尝试运行和修改你的每一个例子。

You just saw how a cell that outputs a table looks inside the book. Here is an example of a cell that outputs text:

如书里面的样子，你刚看了一个单元格如何输出一个表，这里有一个单元输出实例：

```
1+1
```

2

Jupyter will always print or show the result of the last line (if there is one). For instance, here is an example of a cell that outputs an image:

Jupyter会一直打印或显示最后一行的输出结果（如果有一个的话）。例如，这里有个单元格输出一张图片的例子：

```
img = PILImage.create('images/chapter1_cat_example.jpg')
img.to_thumb(192)
```

图：一个猫输出的例子

End sidebar

侧边栏结尾

So, how do we know if this model is any good? In the last column of the table you can see the error rate, which is the proportion of images that were incorrectly identified. The error rate serves as our metric—our measure of model quality, chosen to be intuitive and comprehensible. As you can see, the model is nearly perfect, even though the training time was only a few seconds (not including the one-time downloading of the dataset and the pretrained model). In fact, the accuracy you've achieved already is far better than anybody had ever achieved just 10 years ago!

那么我们如何知道这个模型是否是好的呢？在表格的最后一列你能够看到错误率，这是识别图片正确率的部分。错误率用于我们测量模型质量的直观和易于理解的手段。正如你看到的这个模型近乎完美，即使训练时间只有短短的几秒钟（这个训练过程没有包含一次性下载的数据集和预训练模型）。事实上，你完成的这个精确率已经远好于 10 年前任何人的成就。

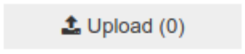
Finally, let's check that this model actually works. Go and get a photo of a dog, or a cat; if you don't have one handy, just search Google Images and download an image that you find there. Now execute the cell with `uploader` defined. It will output a button you can click, so you can select the image you want to classify:

最后让我们检查这个模型的实际工作效果。去获取一张狗或猫的图片；如果你手头上没有，可以通过 Google 图片搜索功能去找并下载你你认为合适的图片。现在在单元格中执行 `uploader` 命令，它会输出一个你能够点击的按钮，你能够选择你想分类的图片：

```
#hide_output
uploader = widgets.FileUpload()
uploader
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

一个 Jupyter 部件不能够被显示，因为部件状态不能被发现。如果内核存储的部件不再有效或部件状态没有在笔记中保存这种情况就会发生。你可通过运行一个合适的单元格能够创建一个部件。



图：一个上传按钮例子

Now you can pass the uploaded file to the model. Make sure that it is a clear photo of a single dog or a cat, and not a line drawing, cartoon, or similar. The notebook will tell you whether it thinks it is a dog or a cat, and how confident it is. Hopefully, you'll find that your model did a great job:

现在你能够通过点击上传按钮上传文件给模型。确保上传的图片只有一只狗或猫的清晰图片，并且不是素描、卡通或其它类似的内容。Jupyter笔记本将会告诉你上传的图片它认为是否是一个狗或猫，以及如何确认它的结果。令人高兴的是，你将会发现你的模型做的非常好：

```
#hide
# For the book, we can't actually click an upload button, so we fake it
uploader = SimpleNamespace(data = ['images/chapter1_cat_example.jpg'])
img = PILImage.create(uploader.data[0])
is_cat,_,probs = learn.predict(img)
print(f"Is this a cat?: {is_cat}.")
print(f"Probability it's a cat: {probs[1].item():.6f}")
```

```
Is this a cat?: True.
这是一只猫吗？  是。
Probability it's a cat: 0.999986
它是一只猫的概率：0.999986
```

Congratulations on your first classifier!

庆祝你的第一个分类器！

But what does this mean? What did you actually do? In order to explain this, let's zoom out again to take in the big picture.

但这意味着什么？你到底做了什么？为了解释这一切，让我们再次缩小一下以看清整体。

What Is Machine Learning?

机器学习是什么?

Your classifier is a deep learning model. As was already mentioned, deep learning models use neural networks, which originally date from the 1950s and have become powerful very recently thanks to recent advancements.

你的分类器是一个深度学习模型。上面已经提到过，深度学习模型使用的起源于 20 世纪 50 年代的神经网络，并且感谢于最近的发展当前它已经变的相当具有影响力。

Another key piece of context is that deep learning is just a modern area in the more general discipline of *machine learning*. To understand the essence of what you did when you trained your own classification model, you don't need to understand deep learning. It is enough to see how your model and your training process are examples of the concepts that apply to machine learning in general.

别一个关键内容部分，深度学习在机器学习的绝大多数学科中只是一个现代领域。当训练你自己的分类模型时你不需要理解深度学习，但要去理解你做的工作的本质，去看你的模型和训练过程通常情况下是如何应用到机器学习的概念例子就足够了。

So in this section, we will describe what machine learning is. We will look at the key concepts, and show how they can be traced back to the original essay that introduced them.

所以在这一部分，我们会描述机器学习是什么。我们将会看到关键概念和展示如何他们能够回溯到那些引出他们的原始文章。

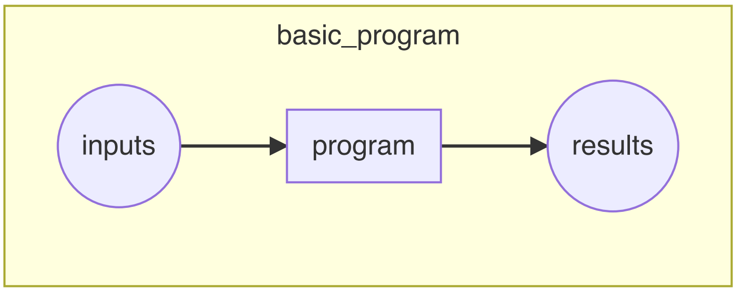
Machine learning is, like regular programming, a way to get computers to complete a specific task. But how would we use regular programming to do what we just did in the last section: recognize dogs versus cats in photos? We would have to write down for the computer the exact steps necessary to complete the task.

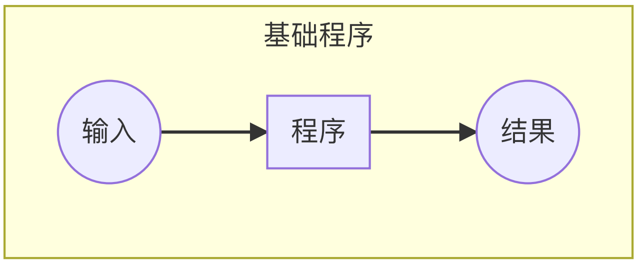
就常规程序一样，机器学习是一种让计算机去完成一个特定任务的方法。但我们如何使用常规程序去做类似上一部门我们刚刚做的事情：在图片中识别狗和猫？我们不得不为计算机写下确切的步骤去完成这个任务。

Normally, it's easy enough for us to write down the steps to complete a task when we're writing a program. We just think about the steps we'd take if we had to do the task by hand, and then we translate them into code. For instance, we can write a function that sorts a list. In general, we'd write a function that looks something like <basic_program> (where *inputs* might be an unsorted list, and *results* a sorted list).

正常情况下，在我们编写一个程序时，对我们来说很容易写下完成一个任务的步骤。我们只用思考如果我们必须通过手工做这个任务的步骤就可以了。例如，我们能够写一个列表分类函数。通过来说，我们写的函数就像下面流程图<基础程序>显示的样子（*输入*一个未分类列表，和输出一个分类*结果*列表）

```
#hide_input
#caption A traditional program
#id basic_program
#alt Pipeline inputs, program, results
gv('''program[shape=box3d width=1 height=0.7]
inputs->program->results''')
```





But for recognizing objects in a photo that's a bit tricky; what *are* the steps we take when we recognize an object in a picture? We really don't know, since it all happens in our brain without us being consciously aware of it!

但对于在图片中做目标识别就有点小困难了；当我们在图片中识别一个目标我们应该采取的步骤有哪些？事实上我们并不知道，因为不用有意识的思考它已经在我们大脑中做完了。

Right back at the dawn of computing, in 1949, an IBM researcher named Arthur Samuel started working on a different way to get computers to complete tasks, which he called *machine learning*. In his classic 1962 essay "Artificial Intelligence: A Frontier of Automation", he wrote:

返回到计算初期，在 1949 年，一个叫亚瑟·塞缪尔的研究员开始了一个不同的任务：让计算机去完成任务，他称为*机器学习*。在他 1962 年发表的“人工智能：自动化前沿”论文中写到：

> : Programming a computer for such computations is, at best, a difficult task, not primarily because of any inherent complexity in the computer itself but, rather, because of the need to spell out every minute step of the process in the most exasperating detail. Computers, as any programmer will tell you, are giant morons, not giant brains.
>
> ：安排计算机做类似计算的是一个困难的任务，主要不是因为计算机自身内在的复杂性，而因为需要在最令人恼火的细节中讲清楚处理的每一个微小步骤。任务一个程序员都会告诉你，计算机不是天才，而是大笨蛋。

His basic idea was this: instead of telling the computer the exact steps required to solve a problem, show it examples of the problem to solve, and let it figure out how to solve it itself. This turned out to be very effective: by 1961 his checkers-playing program had learned so much that it beat the Connecticut state champion! Here's how he described his idea (from the same essay as above):

他基本的思想是：与其告诉计算机解决问题所需要的精确的每一步，不如给它展示一个所要解决这个问题的例子，并让它想出如何自己解决这个问题。这种生产是非常有效率的：在 1961 年，他的跳棋程序已经学了很多，获得了康涅狄格州的冠军！下面描述了他的想法是什么（内容来自上述的同一篇论文）

> : Suppose we arrange for some automatic means of testing the effectiveness of any current weight assignment in terms of actual performance and provide a mechanism for altering the weight assignment so as to maximize the performance. We need not go into the details of such a procedure to see that it could be made entirely automatic and to see that a machine so programmed would "learn" from its experience.
>
> ：假设我们安排了一些自动化工具依据实际表现测试当前权重分配的有效性，并提供一种机制改变权重分配以达到最大化的执行效果。我们不需要进入这个程序的细节去看它是如何实现的全自动化，而是看这样编程的机器将从它的经验中“学习”。

There are a number of powerful concepts embedded in this short statement:

在这个短文中包含了一系列核心概念：

- The idea of a "weight assignment"
- The fact that every weight assignment has some "actual performance"
- The requirement that there be an "automatic means" of testing that performance,
- The need for a "mechanism" (i.e., another automatic process) for improving the performance by changing the weight assignments
- 一个“权重分配”的想法
- 每个权重分配对“实际执行”有一些影响
- 这里有一个测量执行的“自动化工具”需求
- 需要一个“机制”（即：别一个自动化工序）通过改变权重分配以优化执行

Let us take these concepts one by one, in order to understand how they fit together in practice. First, we need to understand what Samuel means by a *weight assignment*.

让我们一个个的对这些概念进行理解，以弄懂实际上他们是怎么有效的聚合在一起的。首先，我们需要去理解塞缪尔对一个*权重分配*的意思。

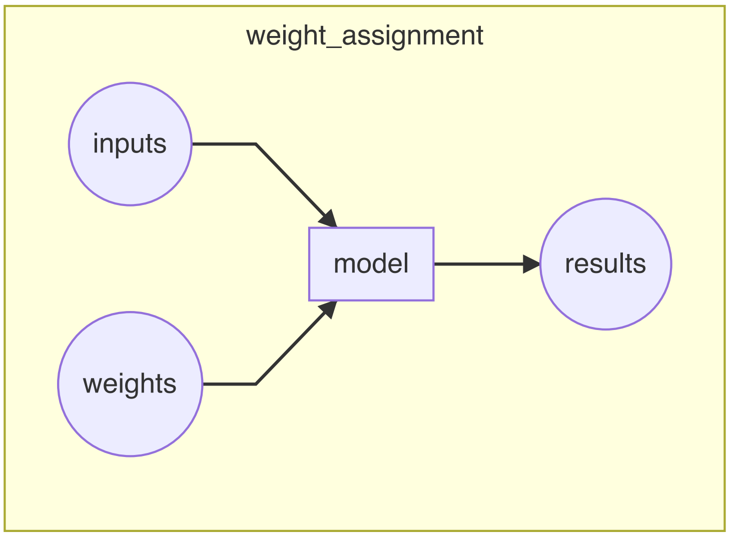
Weights are just variables, and a weight assignment is a particular choice of values for those variables. The program's inputs are values that it processes in order to produce its results—for instance, taking image pixels as inputs, and returning the classification "dog" as a result. The program's weight assignments are other values that define how the program will operate.

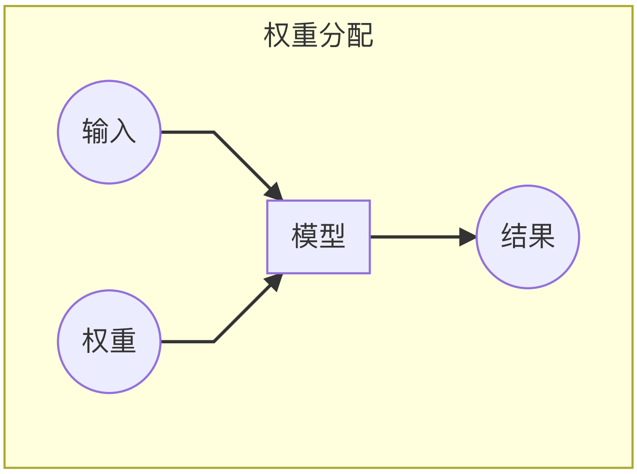
权重只是变量，一个权重是那些变量一个特定选择的值。程序输入的是变量，对它处理以产生结果。例如，以图片像素做为输入，然后返回狗的分类做为结果。程序的权重分配是定义程序将如何进行操作的其它值。

Since they will affect the program they are in a sense another kind of input, so we will update our basic picture in <basic_program> and replace it with <weight_assignment> in order to take this into account.

因此他们将会影响程序，在某种程序上他们是其它类型的输入。考虑到这一点，所以我们会在<基础程序>中更新我们基础描述并把它替换为<权重分配>

```
#hide_input
#caption A program using weight assignment
#id weight_assignment
gv('''model[shape=box3d width=1 height=0.7]
inputs->model->results; weights->model''')
```





We've changed the name of our box from *program* to *model*. This is to follow modern terminology and to reflect that the *model* is a special kind of program: it's one that can do *many different things*, depending on the *weights*. It can be implemented in many different ways. For instance, in Samuel's checkers program, different values of the weights would result in different checkers-playing strategies.

我们已经把小盒子的名字从*程序*变为了*模型*。这沿用的是现代术语和参考的*模型*是一个特殊的程序：它依赖*权重*能够做很多不同的事情。它能以许多不同的方法实施。例如，在塞缪尔的跳棋程序，不同权重值将会在不同的跳棋策略中有不同的结果。

(By the way, what Samuel called "weights" are most generally refered to as model *parameters* these days, in case you have encountered that term. The term *weights* is reserved for a particular type of model parameter.)

（顺便说一下，如果你已经碰到那个术语，塞缪尔命名的“权重”最普遍是现在称为的模型*参数*。术语*权重*保留了模型参数的一个特指类型）

Next, Samuel said we need an *automatic means of testing the effectiveness of any current weight assignment in terms of actual performance*. In the case of his checkers program, the "actual performance" of a model would be how well it plays. And you could automatically test the performance of two models by setting them to play against each other, and seeing which one usually wins.

接下来，塞缪尔说我们需要一个*自动工具依据实际表现测试当前权重分配的有效性*。在他跳棋程序例子中，一个模型的*实际表现*将决定它玩的多么好。你能够通过它们相互间打比赛自动化测试两个模型的性能，并看哪一个会常胜。

Finally, he says we need *a mechanism for altering the weight assignment so as to maximize the performance*. For instance, we could look at the difference in weights between the winning model and the losing model, and adjust the weights a little further in the winning direction.

最后，他说到我们需要一个*改变权重分配的机制以便性能最大化*。例如，你能够看在战胜模型和失败模型之间权重方面的差异，并向胜利的方向进一步调整权重。

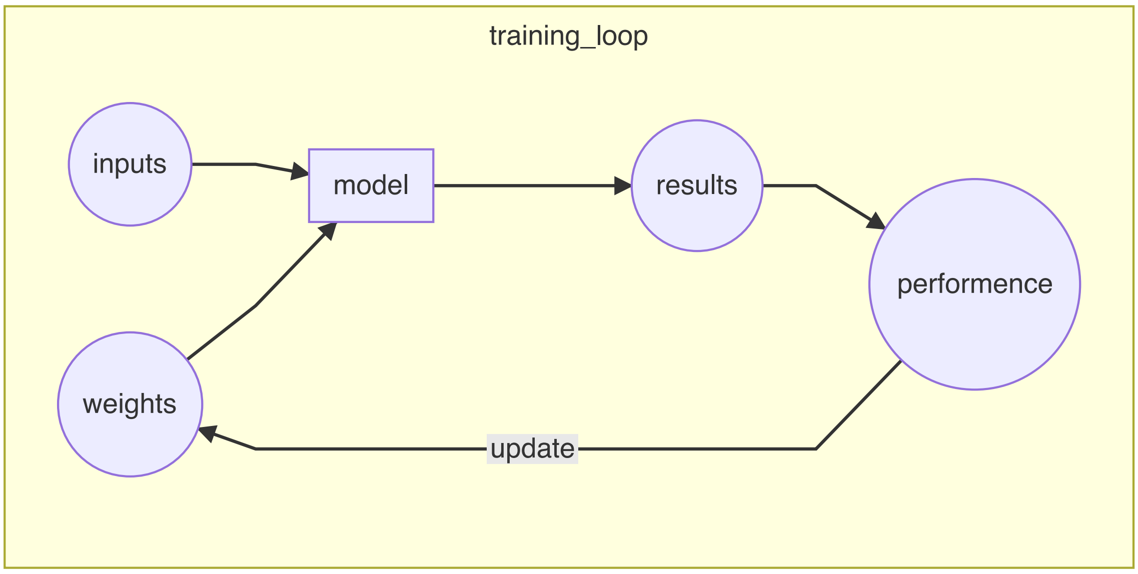
We can now see why he said that such a procedure *could be made entirely automatic and... a machine so programmed would "learn" from its experience*. Learning would become entirely automatic when the adjustment of the weights was also automatic—when instead of us improving a model by adjusting its weights manually, we relied on an automated mechanism that produced adjustments based on performance.

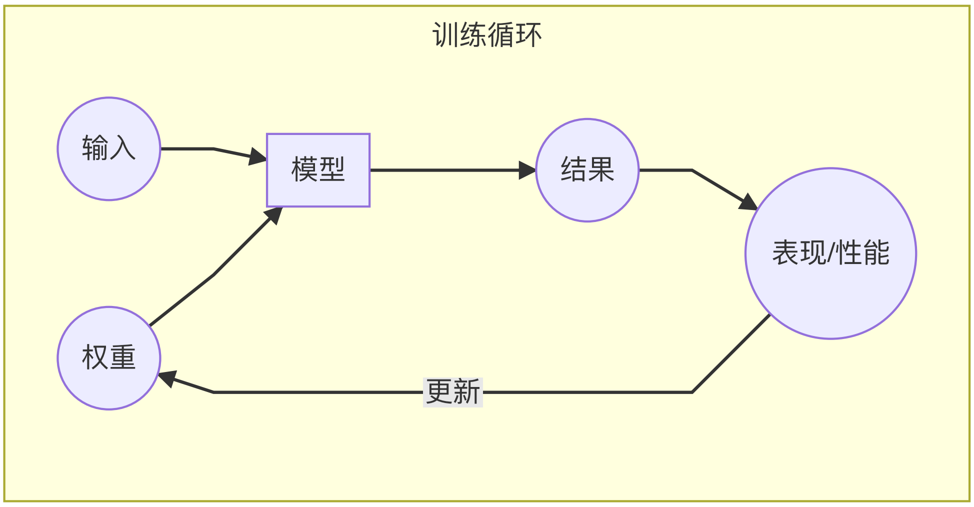
你现在能够明白为什么他说这么一个程序：*能够实现完全的自动化并且...这种程序的机器将从它的经验中“学习”*。当权重调整是这样的自动化（这种自动化是当我们依赖一个自动化机制基于表现产生调整，替代通过手工调整权重改善模型的时候）学习将变的全自动。

<training_loop> shows the full picture of Samuel's idea of training a machine learning model.

下图<训练循环>展示了塞缪尔对于机器学习模型想法的全景图。

```
#hide_input
#caption Training a machine learning model
#id training_loop
#alt The basic training loop
gv('''ordering=in
model[shape=box3d width=1 height=0.7]
inputs->model->results; weights->model; results->performance
performance->weights[constraint=false label=update]''')
```





Notice the distinction between the model's *results* (e.g., the moves in a checkers game) and its *performance* (e.g., whether it wins the game, or how quickly it wins).

注意模型*结果*（即：跳棋走的第一步）和它的*表现*（即：它是否赢得比赛，或如何快速的胜利）之间的区别。

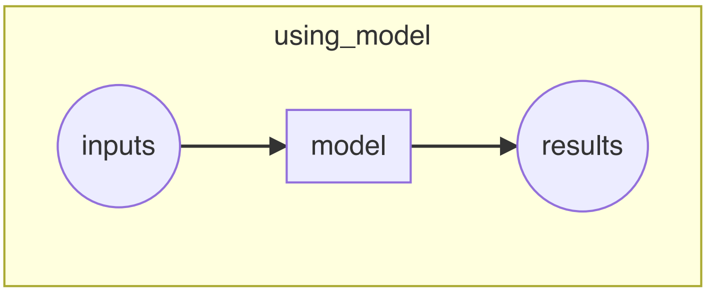
Also note that once the model is trained—that is, once we've chosen our final, best, favorite weight assignment—then we can think of the weights as being *part of the model*, since we're not varying them any more.

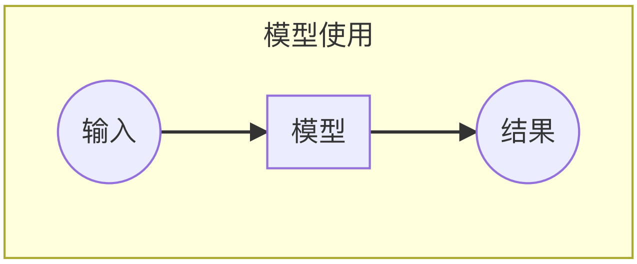
也要注意一旦模型被训练了（一旦我们选择了最终的、最好的、中意的权重分配），我们就会把权重视为模型的一部分，到此为止我们就不会对它进行任何变更了。

Therefore, actually *using* a model after it's trained looks like <using_model>.

因此，实际使用一个被训练后的模型如下图<模型使用>所示。

```
#hide_input
#caption Using a trained model as a program
#id using_model
gv('''model[shape=box3d width=1 height=0.7]
inputs->model->results''')
```





This looks identical to our original diagram in <>, just with the word *program* replaced with *model*. This is an important insight: *a trained model can be treated just like a regular computer program*.

这看起来与我们的原始图是相同的，只是把*程序*替换为*模型*。这是一个重要的点：一个训练的模型能够被看待为像一个普通的计算机程序。

> jargon: Machine Learning: The training of programs developed by allowing a computer to learn from its experience, rather than through manually coding the individual steps.
>
> 行话（黑话）：机器学习：开发训练程序以允许一台计算机从它的经验中进行学习，而不是通过手工写代码具体的每一步。

What Is a Neural Network?

什么是神经网络?

It's not too hard to imagine what the model might look like for a checkers program. There might be a range of checkers strategies encoded, and some kind of search mechanism, and then the weights could vary how strategies are selected, what parts of the board are focused on during a search, and so forth. But it's not at all obvious what the model might look like for an image recognition program, or for understanding text, or for many other interesting problems we might imagine.

把模型想像为可能看起来像一个跳棋程序不是太困难。可能有一系列的跳棋策略的编码和某种搜索机制，然后权重能够改变策略如何被选择，在搜索期间关注于那些可接受的部分，等等。模型可以看起来像一个图像识别程序，或文本理解，或我们可以想像到的其它有趣的问题，模型就不是那么完全清晰了。

What we would like is some kind of function that is so flexible that it could be used to solve any given problem, just by varying its weights. Amazingly enough, this function actually exists! It's the neural network, which we already discussed. That is, if you regard a neural network as a mathematical function, it turns out to be a function which is extremely flexible depending on its weights. A mathematical proof called the *universal approximation theorem* shows that this function can solve any problem to any level of accuracy, in theory. The fact that neural networks are so flexible means that, in practice, they are often a suitable kind of model, and you can focus your effort on the process of training them—that is, of finding good weight assignments.

我们想要某种功能，只通过变化它的权重就能非常灵活的用来解决任何给出的问题。太奇妙了，这种功能事实上是存在的！它就是我们之前讨论过的神经网络。也就是说，如果你把神经网络视为一种数学函数，它就是依赖权重极度灵活的函数。在理论层面，一个数学证明名为*通用近似定理（或万能逼近定理）*显示这个函数能够在任何精度水平解决任何问题。事实上神经网络是一个非常灵活的工作，实际上他们通常是一个合适的模型。你可以聚焦于模型训练过程的效果，即找到一个好的权重分配。

But what about that process? One could imagine that you might need to find a new "mechanism" for automatically updating weight for every problem. This would be laborious. What we'd like here as well is a completely general way to update the weights of a neural network, to make it improve at any given task. Conveniently, this also exists!

但那个过程是什么呢？一种设想是：你可能需要去找一个为每一个问题自动更新权重的新“机制”。这也许是一个艰辛的工作。我们也想要一个完全通用的方式去更新神经网络权重，使它在任何给定的任务中得到完善，这种方式方便的也是存在的！

This is called *stochastic gradient descent* (SGD). We'll see how neural networks and SGD work in detail in <>, as well as explaining the universal approximation theorem. For now, however, we will instead use Samuel's own words: *We need not go into the details of such a procedure to see that it could be made entirely automatic and to see that a machine so programmed would "learn" from its experience.*

这称之为*随机梯度下降*（SGD）。我们在下图会看到神经网络和 SGD 如何工作细节，同时解释通用近似定理。不管怎么样截至目前我们会替换塞缪尔的专用描述：*我们不需要进入这个程序的细节去看它能够实现全自动化，而去看一个这样编程的机器将从它的经验中“学习”*

> J: Don't worry, neither SGD nor neural nets are mathematically complex. Both nearly entirely rely on addition and multiplication to do their work (but they do a *lot* of addition and multiplication!). The main reaction we hear from students when they see the details is: "Is that all it is?"
>
> 杰：不要急，SGD 和神经网络都没有复杂的数学运算。两者几乎完全依赖加法和乘法去做他们的工作（只是它们需要做大量的加法和乘法）。当学生看到细节的时候，我们听到来自他们的主要反馈是：“这就是全部吗？”

In other words, to recap, a neural network is a particular kind of machine learning model, which fits right in to Samuel's original conception. Neural networks are special because they are highly flexible, which means they can solve an unusually wide range of problems just by finding the right weights. This is powerful, because stochastic gradient descent provides us a way to find those weight values automatically.

换句话去总结，一个神经网络是一种特定的正契合塞缪尔原始概念的机器学习模型。神经网络是特别的，因为他们极为灵活，意味着只要通过寻找正确的权重他们能够出乎意料的解决广泛的问题。这很强大，因为随机梯度下降提供给我们去自动找那些权重值的方法。

Having zoomed out, let's now zoom back in and revisit our image classification problem using Samuel's framework.

缩小后，让我们现在再放大并重回我们使用塞缪尔架构的图像分类问题。

Our inputs are the images. Our weights are the weights in the neural net. Our model is a neural net. Our results are the values that are calculated by the neural net, like "dog" or "cat."

我们输入的是图片集。我们的权重是神经网络里的权重。我们的模型是一个神经网络。我们的结果是通过神经网络计算后的值---类似“狗”或“猫”。

What about the next piece, an *automatic means of testing the effectiveness of any current weight assignment in terms of actual performance*? Determining "actual performance" is easy enough: we can simply define our model's performance as its accuracy at predicting the correct answers.

下一部分关于*依据实际表现测试当前权重分配效果的一个自动化工具*是什么？确定“实际表现”是非常容易的：我们能够以它在预测正确答案的精确率简单定义我们模型的表现。

Putting this all together, and assuming that SGD is our mechanism for updating the weight assignments, we can see how our image classifier is a machine learning model, much like Samuel envisioned.

综上所述，假定 SGD 是我们更新权重分配的机制，能看到我们图片分类器是一个更像塞缪尔想像的那种机器学习模型。

A Bit of Deep Learning Jargon

一点深度学习术语

Samuel was working in the 1960s, and since then terminology has changed. Here is the modern deep learning terminology for all the pieces we have discussed:

塞缪尔的工作在 20 世纪 60 年代，从那之后术语已经变了。这里有我们已经讨论过的模型深度学习术语的所有部分：

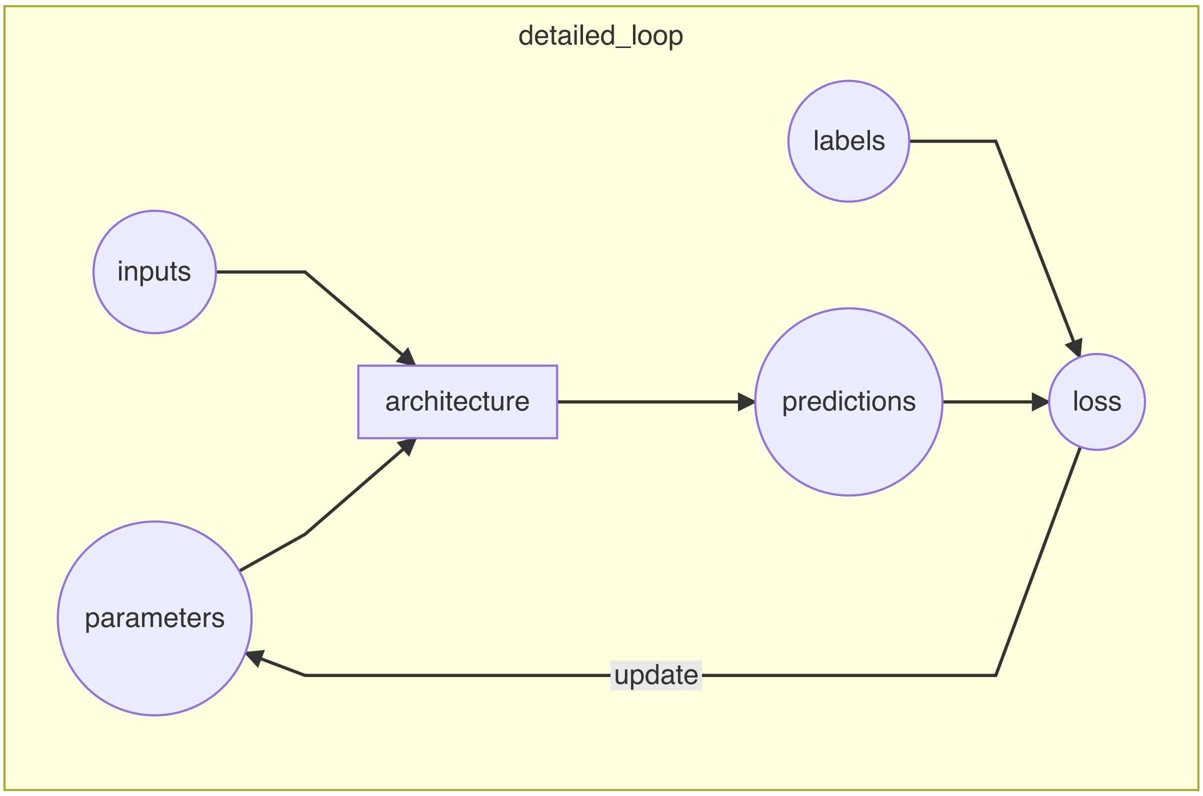
- The functional form of the *model* is called its *architecture* (but be careful—sometimes people use *model* as a synonym of *architecture*, so this can get confusing).
- The *weights* are called *parameters*.
- The *predictions* are calculated from the *independent variable*, which is the *data* not including the *labels*.
- The *results* of the model are called *predictions*.
- The measure of *performance* is called the *loss*.

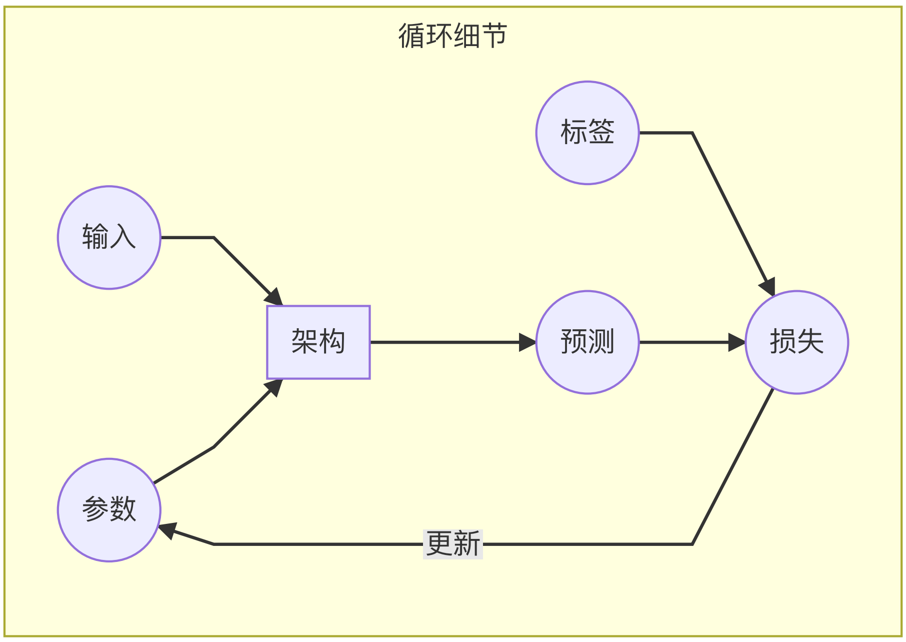
- The loss depends not only on the predictions, but also the correct *labels* (also known as *targets* or the *dependent variable*); e.g., "dog" or "cat."
- *模型*的功能样式称之为*架构*（但请注意，有时候人家会用*模型*做为*架构*的同义词，所以会产生混淆）。
- *权重*被称为*参数*。
- *预测*是由那些不含*标签值数据的自变量*计算的。
- 模型的*结果*称为*预测*。
- *性能*的测量称为*损失*。
- 损失不仅取决于预测，也取决于正确的*标签*（也叫*目标*或*因变量*）；例如“狗”或“猫”。

After making these changes, our diagram in <training_loop> looks like <detailed_loop>.

做了这些改变后，我们的在<训练循环>中的流程图看起来就像下图<循环细节>。

```
#hide_input
#caption Detailed training loop
#id detailed_loop
gv('''ordering=in
model[shape=box3d width=1 height=0.7 label=architecture]
inputs->model->predictions; parameters->model; labels->loss; predictions->loss
loss->parameters[constraint=false label=update]''')
```





Limitations Inherent To Machine Learning

机器学习固有局限

From this picture we can now see some fundamental things about training a deep learning model:

从这个图片我们能够看到一些关于训练一个深度学习模型的基础要素：

- A model cannot be created without data.
- A model can only learn to operate on the patterns seen in the input data used to train it.
- This learning approach only creates *predictions*, not recommended *actions*.
- It's not enough to just have examples of input data; we need *labels* for that data too (e.g., pictures of dogs and cats aren't enough to train a model; we need a label for each one, saying which ones are dogs, and which are cats).
- 模型不能在没有数据的情况下被创建。
- 一个模型只能学习去操作那些用于训练它而输入的数据中所看到的模式。
- 这个学习方法只能创建*预测*，不能推荐*活动*。
- 只有输入数据样例是不够的，我们还需要对那些数据*标注*（例如：狗和猫的图片集不能满足训练一个模式，我们需要对每一张图片进行标注，区分出哪些是狗哪些是猫）

Generally speaking, we've seen that most organizations that say they don't have enough data, actually mean they don't have enough *labeled* data. If any organization is interested in doing something in practice with a model, then presumably they have some inputs they plan to run their model against. And presumably they've been doing that some other way for a while (e.g., manually, or with some heuristic program), so they have data from those processes! For instance, a radiology practice will almost certainly have an archive of medical scans (since they need to be able to check how their patients are progressing over time), but those scans may not have structured labels containing a list of diagnoses or interventions (since radiologists generally create free-text natural language reports, not structured data). We'll be discussing labeling approaches a lot in this book, because it's such an important issue in practice.

通常来说，我们看到大多数组织说他们没有足够的数据，真实的意思是他们没有足够的标注数据。如果有些组织有兴趣在实践中利用模型做一些事情，然后大概他们有一些输入集计划去运行模型来处理。同时大概他们用一些其它方法已经在做了。（例如，手工或一些启发式程序），所以通过那些过程他们已经有数据了！例如，几乎可以确定一个放射科会有医学影像档案（因为他们需要能够检查他们病人随着时间的推移怎样进展的），但是那些影像可能没有包含了诊断和干预列表的结构化标签（因为放射学家通常会写自然语言自由文本报告，而不是结构化数据）。在本书中我们会讨论一些标注方法，因为在实践中它是非常重要的问题。

Since these kinds of machine learning models can only make *predictions* (i.e., attempt to replicate labels), this can result in a significant gap between organizational goals and model capabilities. For instance, in this book you'll learn how to create a *recommendation system* that can predict what products a user might purchase. This is often used in e-commerce, such as to customize products shown on a home page by showing the highest-ranked items. But such a model is generally created by looking at a user and their buying history (*inputs*) and what they went on to buy or look at (*labels*), which means that the model is likely to tell you about products the user already has or already knows about, rather than new products that they are most likely to be interested in hearing about. That's very different to what, say, an expert at your local bookseller might do, where they ask questions to figure out your taste, and then tell you about authors or series that you've never heard of before.

因为这些类型的机器学习模型只能做预测（即尝试复制标签），这能导致在组织目标和模型能力间产品巨大的差异。例如，在本书你会学到怎么创建一个能预测哪些产品用户有可能购买的*推荐系统*。这经常用于电商，这样通过展示最高排名的条目来自定义产品显示在首页。但是这样一个模型通常是通过看用户他们购买历史（*输入*）和他们想要购买的商品或看了什么商品（*标签*）来创建的，这意味着模型可能告诉你关于用户已经购买或已经知道的产品，而不是他们极可能感兴趣于听到的那些全新的产品。这是非常不同的，好比说一个专家在你当地书店可能做的事情，他们问询问问题计算出你的偏好，然后告诉你以前你从未听过的作者或图书系列。

Another critical insight comes from considering how a model interacts with its environment. This can create *feedback loops*, as described here:

其它的评判来自一个模型如何与它的环境进行交互的思考。能够创建*反馈循环*，如下描述：

- A *predictive policing* model is created based on where arrests have been made in the past. In practice, this is not actually predicting crime, but rather predicting arrests, and is therefore partially simply reflecting biases in existing policing processes.
- Law enforcement officers then might use that model to decide where to focus their police activity, resulting in increased arrests in those areas.
- Data on these additional arrests would then be fed back in to retrain future versions of the model.
- 一个*治安预测* 模型是基于在过去某地已经发生的拘捕情况来创建的。事实上，这不是实际的犯罪预测，而是拘捕预测，因此一定程序上会简单的思考方面在已经存在的治安传票上。
- 法律执行官员可能会使用模型去决策他们的警察活动聚焦的区域，结果是在增加拘捕的那些区域。
- 增加拘捕的数据将会被反馈给再训练的未来模型版本。

This is a *positive feedback loop*, where the more the model is used, the more biased the data becomes, making the model even more biased, and so forth.

这是一个*正向的反馈循环*，更多的模型被使用，更多偏好数据产生，使得模型更加倾向偏好，等等。

Feedback loops can also create problems in commercial settings. For instance, a video recommendation system might be biased toward recommending content consumed by the biggest watchers of video (e.g., conspiracy theorists and extremists tend to watch more online video content than the average), resulting in those users increasing their video consumption, resulting in more of those kinds of videos being recommended. We'll consider this topic more in detail in <chapter_ethics>.

在商业环境中反馈循环也能产生一些问题。例如，一个视频推荐系统可能偏好推荐观看量最大的内容（例如，阴谋论和极端分子相比普通人倾向观看更多的在线视频），导致那些用户增加他们的视频观看量，进而导致更多那种类型的视频被推荐。我们会在<伦理章节>思考这个话题的更多细节。

Now that you have seen the base of the theory, let's go back to our code example and see in detail how the code corresponds to the process we just described.

现在你已经看到了这个理论基础，让我们返回到代码实例，并看代码如何反映到我们刚刚描述的过程细节。

How Our Image Recognizer Works

我们的图像识别如何工作

Let's see just how our image recognizer code maps to these ideas. We'll put each line into a separate cell, and look at what each one is doing (we won't explain every detail of every parameter yet, but will give a description of the important bits; full details will come later in the book).

The first line imports all of the fastai.vision library.

```
from fastai.vision.all import *
```

This gives us all of the functions and classes we will need to create a wide variety of computer vision models.

> J: A lot of Python coders recommend avoiding importing a whole library like this (using the `import *` syntax), because in large software projects it can cause problems. However, for interactive work such as in a Jupyter notebook, it works great. The fastai library is specially designed to support this kind of interactive use, and it will only import the necessary pieces into your environment.