

Your Deep Learning Journey

あなたのディープラーニング・ジャーニー

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 many people as possible mathematical background (though it's okay if you do too!); we wrote this book to make deep learning accessible to as

こんにちは、そして、あなたのディープラーニングの旅に、どんなに遠くても、私たちを参加させてくれてありがとう！この章では、本書の内容をもう少し詳しく説明し、深層学習の背後にある重要な概念を紹介し、さまざまなタスクで最初のモデルを訓練します。技術や数学のバックグラウンドがなくても問題ありません（なくても大丈夫です！）。

Deep Learning Is for Everyone

ディープラーニングはみんなのもの

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. <> is a list of a few things you *absolutely don't need* to do world-class deep learning.

多くの人が、ディープラーニングで素晴らしい結果を得るために、いろいろと難しいものが必要だと思い込んでいますが、本書でわかるように、そうした人たちは間違っています。<>は、世界レベルのディープラーニングを行うために絶対に必要なものをいくつか挙げたリストです。

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.

ディープラーニングとは、人間の音声認識から動物の画像分類まで、さまざまなデータを抽出・変換するコンピュータ技術であり、複数の層のニューラルネットワークを使用することで実現します。各層は、前の層からの入力を受けて、徐々に洗練されていきます。各層は、誤差を最小限に抑え、精度を向上させるアルゴリズムで学習されます。このようにして、ネットワークは指定されたタスクの実行を学習する。学習アルゴリズムについては、次のセクションで詳しく説明します。

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](#).

ディープラーニングには、パワーと柔軟性、そしてシンプルさがあります。そのため、多くの分野で応用されるべきだと考えています。社会科学や物理科学、芸術、医学、金融、科学研究など、さまざまな分野です。個人的な例を挙げると、ジェレミーは医学のバックグラウンドがないにもかかわらず、ディープラーニングのアルゴリズムを使って病気や疾患を診断する会社、Enlitic を立ち上げました。起業から数カ月で、同社のアルゴリズムが放射線科医よりも正確に悪性腫瘍を特定できることが発表されました。

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:

ディープラーニング、あるいはディープラーニングを多用した手法が、今や世界一となった様々な分野の何千ものタスクの一部を紹介します：

- Natural language processing (NLP):: Answering questions; speech recognition; summarizing documents; classifying documents; finding names, dates, etc. in documents; searching for articles mentioning a concept

自然言語処理（NLP）：： 質問に対する回答、音声認識、文書の要約、文書の分類、文書中の名前、日付などの検索、ある概念に言及した記事の検索

- 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

コンピュータビジョン：： 衛星やドローンによる画像解釈（災害対策など）、顔認識、画像キャプション、交通標識の読み取り、自律走行車での歩行者や車両の位置確認など

- 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：： CT、MRI、X線などの放射線画像の異常発見、病理スライドの特徴のカウント、超音波の特徴の測定、糖尿病性網膜症の診断など。

- 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：： タンパク質の折り畳み、タンパク質の分類、腫瘍と正常な配列の決定や臨床的に有効な遺伝子変異の分類などの多くのゲノミクス作業、細胞の分類、タンパク質とタンパク質の相互作用の分析

- Image generation:: Colorizing images; increasing image resolution; removing noise from images; converting images to art in the style of famous artists

画像生成: : 画像のカラー化、画像の高解像度化、画像のノイズ除去、有名アーティスト風のアートへの変換

- Recommendation systems:: Web search; product recommendations; home page layout

レコメンデーションシステム: : ウェブ検索、商品推薦、ホームページのレイアウト

- Playing games:: Chess, Go, most Atari video games, and many real-time strategy games

ゲームをする: : チェス、囲碁、アタリ社のビデオゲーム全般、リアルタイムストラテジーゲーム全般

- Robotics:: Handling objects that are challenging to locate (e.g., transparent, shiny, lacking texture) or hard to pick up

ロボティクス: : 見つけるのが難しい（例：透明、光沢がある、質感がない）、または拾いにくい物体の取り扱い

- Other applications:: Financial and logistical forecasting, text to speech, and much more...

その他のアプリケーション: : 金融や物流の予測、テキストからスピーチへ、などなど...。

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.

注目すべきは、ディープラーニングがこれほど多様な応用を持ちながら、ほぼすべてのディープラーニングがニューラルネットワークという1種類のモデルに基づいていることです。

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.

しかし、ニューラルネットワークは、実はまったく新しいものではありません。この分野をより広い視野で見るためには、少し歴史から入ってみる価値があるのです。

Neural Networks: A Brief History

ニューラルネットの 簡単な歴史

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:

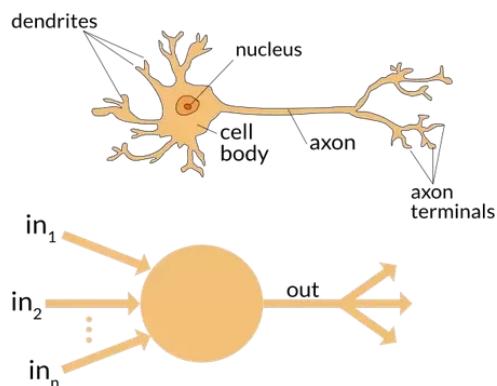
1943年、神経生理学者のウォーレン・マッカロクと論理学者のウォルター・ピツは、人工ニューロンの数学的モデルを開発するために手を組みました。彼らの論文「神経活動に内在する観念の論理的計算」の中で、彼らは次のように宣言した：

: 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.

：神経活動の「全か無か」の性格から、神経の出来事とその間の関係は命題論理によって扱うことができる。すべてのネットの動作は、このような用語で記述できることがわかった。

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.

マッカロクとピツは、<>に示すように、実際の神経細胞の簡略化したモデルが、単純な加算と閾値を使って表現できることに気づいた。ピツは独学で学び、12歳の時には、ケンブリッジ大学で偉大なバートランド・ラッセルのもとで学ぶという誘いを受けていた。しかし、この誘いには乗らず、生涯、学位や権威のある地位の誘いにも乗らなかった。彼の有名な仕事のほとんどは、ホームレス時代に行われたものである。公的に認められた地位がなく、社会的孤立を深めていたにもかかわらず、マッカロクとの仕事は影響力を持ち、フランク・ローゼンブルットという心理学者に取り上げられた。



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.

ローゼンブラットは、さらに人工ニューロンを発展させ、学習能力を持たせた。さらに重要なのは、この原理を実際に使った最初の装置、「マーク I パーセプトロン」の製作に取り組んだことだ。ローゼンブラットは『知的オートマトンの設計』の中で、この仕事についてこう書いている：「人間の訓練や制御なしに、周囲の環境を知覚し、認識し、識別することができる機械である。パーセプトロンが作られ、簡単な図形を認識することに成功した。

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.

MIT のマーヴィン・ミンスキーという教授は、ローゼンブラットの発明について、セイモア・パパートとともに『パーセプトロン』(MIT 出版)という本を書きました(ローゼンブラットの高校は同じ学年でした！)。彼らは、この装置の 1 層では、簡単だが重要な数学的機能(XOR など)を学習することができないことを示しました。また、同書では、このデバイスを多層化することで、これらの制限に対処できることも示している。しかし、残念ながら、このうち最初の 1 つだけが広く認識されたにすぎません。その結果、世界の学術界はその後 20 年間、ニューラルネットワークをほぼ完全に見限ってしまった。

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:

過去 50 年間でニューラルネットワークの最も重要な研究は、1986 年に MIT Press から発売された David Rumelhart, James McClellan, および PDP Research Group による複数巻の Parallel Distributed Processing (PDP) であろう。第 1 章では、ローゼンブラットが示したのと同様の希望が示されている：

: 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.

：人が今日のコンピュータより賢いのは、脳が、人が得意とする自然な情報処理タスクの中心的な側面を扱うのに、より適した基本的な計算アーキテクチャを採用しているからである。

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.

PDP がここで使っている前提は、従来のコンピュータプログラムは脳とは全く異なる働きをしており、そのためにコンピュータプログラムは（その時点では）、脳が簡単にできること（絵の中の物体を認識するなど）が苦手だったのではないかというものです。PDP のアプローチは、脳の働き方に「他のフレームワークよりも近い」ので、この種のタスクをよりうまく処理できるかもしれない、と著者たちは主張しました。

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:

実際、PDP のアプローチは、今日のニューラルネットワークで使われているアプローチと非常によく似ている。この本では、並列分散処理について、次のように定義しています：

1. A set of *processing units*

処理装置のセット

2. A *state of activation*

活性化している状態

3. An *output function* for each unit

各ユニットの出力機能

4. A *pattern of connectivity* among units

ユニット間の接続のパターン

5. A *propagation rule* for propagating patterns of activities through the network of connectivities

アクティビティのパターンをコネクティビティのネットワークを通じて伝播させるための伝播ルール

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

あるユニットに与えられた入力とそのユニットの現在の状態を組み合わせて、そのユニットの出力を生成するための活性化ルール。

7. A *learning rule* whereby patterns of connectivity are modified by experience

接続性のパターンが経験によって修正される学習規則

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

システムが動作しなければならない環境

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

本書では、最新のニューラルネットワークがこれらの要件にそれぞれ対応していることを紹介する。

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.

1980年代には、ほとんどのモデルがニューロンの第2層で構築されていたため、ミンスキーとパパートが指摘した問題（これは、上記の枠組みを使えば、彼らの「ユニット間の接続性のパターン」である）を回避することができた。そして実際に、ニューラルネットワークは80年代から90年代にかけて、実際の実用的なプロジェクトに広く利用されるようになりました。しかし、ここでもまた、理論的な問題に対する誤解が、この分野の足かせとなつた。理論的には、ニューロンを1層追加するだけで、どんな数学的関数もニューラルネットワークで近似できるのだが、実際にはそのようなネットワークは大きすぎ、遅すぎて使い物にならないことが多い。

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."

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...

30年前に、実用的な性能を得るためにには、さらに多くのニューロン層を使用する必要があることが研究者によって示されました。この原則がより広く理解され、適用されるようになったのは、ここ10年ほどのことです。コンピュータのハードウェアの改良、データの利用可能性の向上、そしてニューラルネットワークをより速く簡単に学習させるアルゴリズムの改良により、より多くの層を使用することができるようになり、その結果、ニューラルネットワークは今ようやくその潜在能力を発揮しています。私たちは今、ローゼンブルattが約束したものを持ちました：「人間の訓練や制御なしに、周囲の環境を知覚し、認識し、識別できる機械」。

これが、本書であなたが作る方法を学ぶものです。しかし、その前に、これから長い時間を一緒に過ごすのだから、お互いのことを少し知っておこう....。

Who We Are

私たちとは

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

私たちは、この旅の案内人であるシルヴァンとジェレミーです。私たちがこのポジションに適していることをご理解いただけすると幸いです。

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.

ジェレミーは、30年ほど前から機械学習を使い、教えています。彼がニューラルネットワークを使い始めたのは 25 年前です。この間、ディープラーニングと医療に焦点を当てた最初の会社 Enlitic の設立や、世界最大の機械学習コミュニティ Kaggle の社長兼チーフサイエンティストを務めるなど、機械学習を中心とする多くの会社やプロジェクトを率いてきました。本書のベースとなるコースを構築した fast.ai では、レイチェル・トマス博士とともに共同創業者でもある。

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

時折、ジェレミーのこのようなサイドバーで、私たちから直接話を聞くことができます：

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...

J: みなさんこんにちは、ジェレミーです！私はジェレミーです！私が正式な技術教育を受けていないことを、皆さんはご存知でしょうか。哲学を専攻して学士号を取得しましたが、成績はあまりよくありませんでした。理論的

な勉強よりも、実際のプロジェクトを行うことに興味があったので、大学時代はマッキンゼー・アンド・カンパニーという経営コンサルティング会社でフルタイムで働きました。もしあなたが、抽象的な概念を何年もかけて学ぶよりも、実際に手を動かして物を作る方が好きな人なら、私の考え方方が理解できるはずです！私のような数学的、あるいは正式な技術的背景を持たない人たちに最適な情報を提供するために、私のサイドバーを見てください…

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!

一方、シルヴァンは、正式な技術教育についてよく知っています。実際、彼は数学の教科書を10冊書いていて、フランスの数学の上級カリキュラムをすべてカバーしています！

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…

S: ジェレミーと違って、私は機械学習アルゴリズムのコーディングや応用に何年も費やしてきたわけではありません。むしろ、ジェレミーのfast.aiコースのビデオを見て、最近、機械学習の世界に足を踏み入れたのです。ですから、もしあなたがターミナルを開いてコマンドラインでコマンドを書いたことがない人なら、私がどこから来たのか理解していただけると思います！私のサイドバーには、より数学的で正式な技術的背景を持ち、実際のコーディング経験が少ない人、つまり私のような人に最適な情報が掲載されていますので、ぜひご覧ください…

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のコースは、世界中のあらゆる階層の何十万人もの生徒が学んでいます。シルヴァンは、ジェレミーが見た中で最も印象的な受講生として際立っており、それがきっかけでfast.aiに参加し、ジェレミーとともにfastaiソフトウェアライブラリの共著者となった。

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.

数学の専門家であり、コーディングと機械学習の専門家でもある。また、数学の分野では比較的部外者であり、コーディングと機械学習の分野では比較的部外者であるという感覚を理解している人たちもある。

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.

スポーツを見たことがある人なら誰でも知っていることですが、2人の解説チームがある場合、「特別コメント」をする3人目が必要です。そのスペシャルコメンテーターがアレクシス・ギャラガーです。アレクシスは、数理生物学の研究者、脚本家、即興パフォーマー、マッキンゼーのコンサルタント（ジェレミーと同じ！）、Swiftのコーダー、CTOと、非常に多様な経験を持っています。

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.

A: そろそろAIについて勉強しようと思っているんだ！でも、機械学習モデルを構築した経験はないんです。でも...そんなに難しいことなんでしょうか？この本では、皆さんと同じように、私も勉強していくつもりです。私のサイドバーには、私の旅で役に立った学習のヒントが掲載されているので、ぜひご覧ください。

How to Learn Deep Learning

ディープラーニングの学び方

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.

『Making Learning Whole』（Jossey-Bass）を書いたハーバード大学のデビッド・パーキンス教授は、教えることについて多くのことを語っている。基本的な考え方は、ゲーム全体を教えるということです。つまり、野球を教えるのであれば、まず野球の試合に連れて行ったり、野球をやらせたりするのです。麻ひもを巻いて野球ボールを作る方法や、放物線の物理学、バットの上のボールの摩擦係数などは、一から教えないのです」。

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.

コロンビア大学の数学博士で、ブラウン大学の元教授、幼稚園から高校までの数学教師であるポール・ロックハートは、影響力のあるエッセイ「A Mathematician's Lament」の中で、音楽や芸術が数学と同じように教えられている悪夢の世界を想像しています。子供たちは、10年以上かけて楽譜と理論を習得し、楽譜を別の調に移調する授業を受けるまで、音楽を聴いたり演奏したりすることは許されない。美術の授業では、色やアプリケーターについて学びますが、実際に絵を描くことは大学まで許されません。ばかばかしいと思われますか？ 数学の授業では、何年もかけて丸暗記をさせ、乾き切った基礎知識を学ばせるのです。

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!

残念ながら、深層学習に関する多くの教材もここから始まっています。学習者には、実際に動作するコードの例を示すことなく、ヘシアンの定義や損失関数のティラー近似の定理に従うよう求めているのです。私たちは微積分を非難しているのではありません。私たちは微積分を愛していますし、シリヴァンも大学レベルで教えていますが、深層学習を学ぶ際に微積分から始めるのがベストだとは思っていません！

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.

ディープラーニングでは、モデルをより良くするために修正するモチベーションがあれば、本当に役に立ちます。そのときこそ、関連する理論を学び始めるときなのです。しかし、そもそもモデルを持っている必要があります。私たちは、ほとんどすべてのことを実例を用いて教えています。その実例を積み重ねていくうちに、どんどん深みにはまっていき、あなたのプロジェクトをより良いものにしていく方法を教えていきます。つまり、必要な理論的基礎はすべて、文脈の中で、なぜそれが重要なのか、どのように機能するのかがわかるような形で、徐々に学んでいくことになるのです。

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

そこで、私たちからあなたへの約束です。本書を通じて、私たちは以下の原則に従います：

- 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...

ゲーム全体を教える。まずは、シンプルで表現力豊かなツールを使って、実世界の問題を解決するために、完全で動作し、非常に使い勝手の良い、最先端のディープラーニングネットワークを使う方法を紹介することから始めます。そして、そのツールがどのように作られているのか、そのツールを作るツールがどのように作られているのか、などなど、徐々に理解を深めていきます。

- 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.

常に例題を通して教える。代数的な記号操作から始めるのではなく、直感的に理解できるような文脈や目的があることを確認するのです。

- Simplifying as much as possible. We've spent years building tools and teaching methods that make previously complex topics very simple.

可能な限り簡略化する。私たちは、これまで複雑だったトピックを非常にシンプルにするためのツールや教授法を長年かけて構築してきました。

- Removing barriers. Deep learning has, until now, been a very exclusive game. We're breaking it open, and ensuring that everyone can play.

障壁を取り除く。ディープラーニングはこれまで、非常に排他的なゲームでした。私たちはそれを打ち破り、誰もがプレイできるようにします。

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.

ディープラーニングで最も難しいのは職人技です。十分なデータがあるかどうか、正しいフォーマットになっているかどうか、モデルが適切にトレーニングされているかどうか、もしそうでないなら、どうすればいいのか、どうやって知ることができるのでしょうか。だからこそ、私たちは「やってみる」ことで学ぶことを大切にしています。データサイエンスの基本的なスキルと同様に、ディープラーニングも実践的な経験を通じてのみ上達します。理論に時間をかけようとすると、逆効果になることもあります。重要なのは、ひたすらコーディングして問題を解決することです。理論は、コンテキストとモチベーションがあるときに、後で考えればいいのです。

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.

旅が辛く感じることもあるでしょう。行き詰まりを感じることもあるでしょう。でも、あきらめないでください！本を巻き戻して、間違いなく行き詰っていない最後の部分を見つけ、そこからゆっくり読み進めて、最初にわからないことを見つけるのです。そして、自分でコードの実験をしてみたり、困っている問題についてもっとチュートリアルがないかググってみたりしてください-多くの場合、その材料について別の角度から見ると、理解しやすくなることがあります。また、一読してすべて（特にコード）を理解できることは、予想されることであり、普通のことです。一通り理解してから先に進もうとすると、難しいこともあります。その先にある部分から、より大きな文脈を得ることで、物事がうまくいくこともあるのです。ですから、もある部分で行き詰まつたら、とにかく先に進んでみて、後でまた戻ってくるようにメモしておいてください。

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](#):

ディープラーニングで成功するためには、特別な学歴は必要ないことを忘れないでください。例えば、「Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks」は、過去10年間で最も影響力のある論文の1つで、5,000回以上引用されていますが、これはアレック・ラドフォードが学部生のときに書いたものです。自動運転車を作るという極めて困難な課題を解決しようとしているテスラでも、CEOのイーロン・マスクがこう言っています：

: 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.

：博士号は絶対に必要ありません。重要なのは、AIに対する深い理解と、実際に役立つ方法でNNを実装する能力だけだ(後者の点こそが本当に難しいことだ)。高校を卒業しているかどうかは気にしないでください。

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.

しかし、成功するために必要なことは、本書で学んだことを個人的なプロジェクトに適用し、常に忍耐強く取り組むことです。

Your Projects and Your Mindset

あなたのプロジェクトとあなたのマインドセット

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.

葉の写真から植物が病気にかかっているかどうかを特定したい、編み物のパターンを自動生成したい、X線写真から結核を診断したい、アライグマが猫用ドアを使用しているときを判断したいなど、ディープラーニングを自分の問題で（他の人から事前に訓練されたモデルを使って）できるだけ早く使えるようにし、徐々に詳細を掘り下げていく予定です。次の章の最初の30分で、ディープラーニングを使って最先端の精度で自分の問題を解決する方法を学ぶことができます！(すぐにでもコーディングを始めたい方は、今すぐにでも読み飛ばしてください)。世の中には、ディープラーニングを行うには、Googleのような規模の

コンピュータリソースとデータセットが必要だという悪質な俗説がありますが、そんなことはありません。

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!

では、どのようなタスクが良いテストケースになるのでしょうか？ピカソの絵とモネの絵を区別するようにモデルを訓練したり、息子の写真ではなく娘の写真を選ぶように訓練したりすることができます。趣味や情熱に集中するのも有効です。大きな壮大な問題を解決しようとするよりも、4つか5つの小さなプロジェクトを自分に課すほうが、始めたばかりのころはうまくいく傾向があります。行き詰まりやすいので、あまりに早くから野心的なことをしようとすると、かえって逆効果になることがあります。そして、基本が身についたら、自分が本当に誇れるものを完成させることを目指しましょう！

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.

J: ディープラーニングは、どんな問題にも対応できますね。例えば、私の最初のスタートアップは FastMail という会社で、1999 年の創業当時は電子メールサービスを強化したものでした（現在もそうです）。2002 年に私は、ディープラーニングの原型である単層ニューラルネットワークを使って、メールを分類し、顧客がスパムを受け取らないようにするための仕組みを作りました。

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.

ディープラーニングが得意な人に共通する性格的特徴として、遊び心や好奇心が挙げられます。物理学者の故リチャード・ファインマンは、ディープラーニングが得意と思われる人物の一例です。彼は、空中で回転する皿の揺れ方を面白がって、素粒子の動きを理解するようになったのです。

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

では、何を学ぶのか、まずはソフトウェアから見ていきましょう。

The Software: PyTorch, fastai, and Jupyter

(And Why It Doesn't Matter)

ソフトウェアのことです: PyTorch、fastai、Jupyter の 3 つ。

(そして、なぜそれが重要でないのか)

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.

私たちは、何十種類ものパッケージと多くの異なるプログラミング言語を使って、何百もの機械学習プロジェクトを完成させてきました。fast.ai では、現在使われている主な深層学習や機械学習のパッケージのほとんどを使ってコースを書いてきました。2017 年に PyTorch が登場した後、私たちは 1000 時間以上かけてテストし、将来のコース、ソフトウェア開発、研究に使用することを決定しました。それ以来、PyTorch は世界で最も急速に成長している深層学習ライブラリとなり、すでにトップカンファレンスでほとんどの研究論文に使用されています。これは一般に、産業界における利用の先行指標となります。なぜなら、これらの論文は最終的に製品やサービスに商業的に利用されるからです。私たちは、PyTorch が深層学習のための最も柔軟で表現力のあるライブラリであることを発見しました。PyTorch は、スピードとシンプルさを引き換えにするのではなく、その両方を提供するものです。

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.

PyTorch は低レベルの基礎ライブラリとして最適で、より高度な機能のための基本操作を提供します。fastai ライブラリは、PyTorch の上にこの高水準の機能を追加するための最も人気のあるライブラリです。また、深く階層化されたソフトウェアアーキテクチャを提供する点でユニークであるため、本書の目的にも特に適しています（この階層化された API については査読付きの学術論文もあります）。本書では、深層学習の基礎をどんどん深めていくとともに、fastai のレイヤーもどんどん深めています。本書では、多くのユニークな機能を提供する一から書き直した fastai ライブラリのバージョン 2 を扱います。

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 clearly 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.

しかし、ライブラリから別のライブラリへの切り替えを学ぶには数日しかからないので、どのソフトウェアを学ぶかはあまり重要ではありません。本当に重要なのは、ディープラーニングの基礎と技術をきちんと学ぶことです。私たちが重視するのは、学ぶべき概念を明確に表現したコードを使うことです。高レベルの概念を教えるところでは、高レベルの fastai コードを使用します。低レベルの概念を教えるところでは、低レベルの PyTorch、あるいは純粋な Python のコードを使用することになります。

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.

最近、新しいディープラーニングライブラリが急ピッチで登場しているように感じるとなったら、今後数ヶ月、数年の間にもっと速いペースで変化することを覚悟する必要があるのではないか？この分野に参入する人が増えれば、彼らはより多くのスキルやアイデアをもたらし、より多くのことを試すようになるでしょう。今日学んだ具体的なライブラリやソフトウェアが何であれ、1~2年後には時代遅れになっていると考えた方がよいでしょう。ディープラーニングよりもはるかに成熟し、成長も遅いウェブプログラミングの世界で、ライブラリや技術スタックの変更が常に行われていることを考えればわかるでしょう。私たちは、学習の焦点は、基礎となる技術を理解し、それを実際にどのように適用するか、そして、新しいツールや技術がリリースされたときに、どのように迅速に専門知識を構築するかに置く必要があると強く信じています。

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.

本書の終わりには、fastai に含まれるほぼすべてのコード（PyTorch の多くも）を理解できるようになります。各章では、モデルを構築して訓練する際に何が起こっているかを正確に示すために、さらに深く掘り下げます。つまり、最新のディープラーニングで使われている最も重要なベストプラクティスを学ぶことができます。ただ単に使い方を学ぶだけでなく、それらが実際にどのように機能し、実装されているのかを知ることができます。もし、そのようなアプローチを他のフレームワークで使いたい場合にも、必要な知識を得ることができます。

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!

ディープラーニングを学ぶ上で最も重要なのはコードを書いて実験することなので、コードで実験するための優れたプラットフォームを持っていることが重要です。最も人気のあるプログラミング実験プラットフォームは、Jupyterと呼ばれるものです。本書では、これを使つて使っていきます。Jupyterを使ってモデルを訓練・実験し、データの前処理とモデル開発のパイプラインの各段階をイントロスペクションする方法を紹介します。Jupyter Notebookは、Pythonでデータサイエンスを行うための最も人気のあるツールですが、これには理由があります。パワフルで柔軟性があり、使いやすいのです。きっと気に入っているだけだと思います！

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

では、実際に使ってみて、最初のモデルをトレーニングしてみましょう。

Your First Model

初めてのモデル

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.

先ほども申し上げたように、「なぜそうなるのか」を説明する前に、「どうやるのか」をお教えします。このトップダウンのアプローチに従って、まず、犬と猫をほぼ100%の精度で認識する画像分類器を実際にトレーニングすることから始めます。このモデルを訓練し、実験を行うには、いくつかの初期設定が必要です。見た目ほど難しくはありませんので、ご安心ください。

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.

s: 特に、ターミナルやコマンドラインのようなものを使った経験がほとんどない場合は、最初は怖くてもセットアップの部分をスキップしないでください。特に、ターミナルやコマンドラインなどの使用経験がほとんどない場合は、そのほとんどが必要ありません。本書と並行して、自分なりの実験を行なうことが、学習する上で非常に重要です。

Getting a GPU Deep Learning Server

GPU ディープラーニングサーバーの入手

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.

本書のほぼすべてを実行するには、NVIDIA GPU を搭載したコンピュータにアクセスする必要があります（残念ながら、他のブランドの GPU は、主な深層学習ライブラリで完全にサポートされていません）。しかし、GPU を購入することはお勧めしませんし、すでに持っている場合でも、まだ使用することはお勧めしません！コンピュータのセットアップには時間とエネルギーが必要であり、今すぐディープラーニングに全力を注ぎたいはずです。そこで、必要なものがすべてプリインストールされたパソコンをレンタルして、すぐに使えるようにすることをお勧めします。費用は、使用している間、1 時間あたり 0.25 米ドル程度で済みますし、無料のオプションもあります。

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.

専門用語 グラフィック・プロセッシング・ユニット (GPU) : グラフィックカードとも呼ばれる。コンピュータに搭載されている特殊なプロセッサで、何千もの単一タスクを同時に処理することができ、特にゲームをプレイするためにコンピュータに 3D 環境を表示するために設計されています。GPU は、ニューラルネットワークを通常の CPU の何百倍もの速さで実行することができます。現代のコンピュータには GPU が搭載されていますが、ディープラーニングに必要な GPU を搭載しているものはほとんどありません。

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.

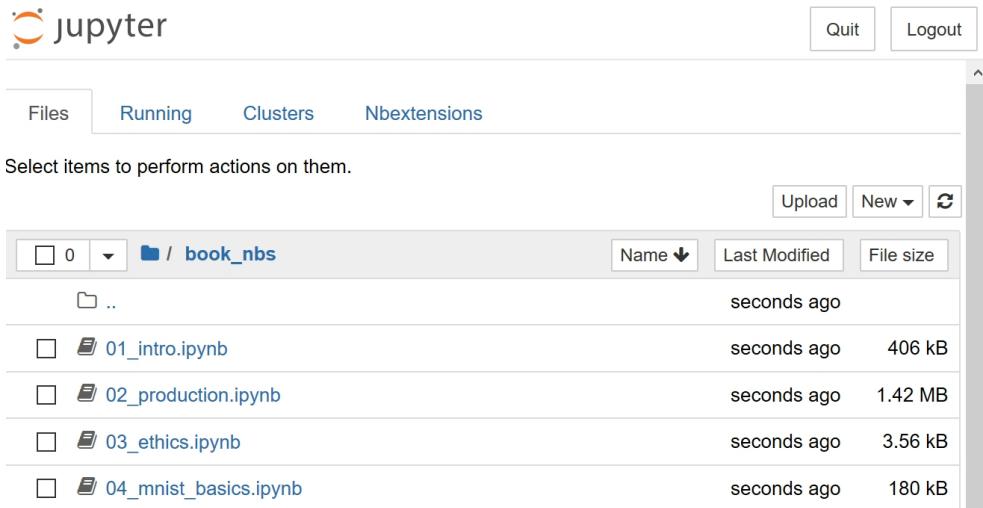
本書で使用する GPU サーバーのベストチョイスは、企業の誕生と価格の変化により、時間の経過とともに変化します。本書のウェブサイトでは、私たちが推奨する選択肢のリストを管理しています。今すぐそこにアクセスして、指示に従って GPU ディープラーニングサーバーに接続しましょう。心配しないでください、ほとんどのプラットフォームでセットアップにかかる時間は 2 分ほどで、多くは支払いやクレジットカードさえも必要ありません。

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.

A: 私の 2 つの意見: このアドバイスに耳を傾けてください! コンピュータが好きなら、自分のボックスを立ち上げたいという誘惑に駆られるでしょう。気をつけましょう! 実現は可能ですが、意外に手間がかかり、気が散ってしまいます。この本のタイトルが「Ubuntu システム管理、NVIDIA ドライバのインストール、apt-get、conda、pip、Jupyter Notebook 設定についてあなたがこれまで知りたかったことすべて」ではないのは、それなりの理由があります。それだけで 1 冊の本になってしまいます。職場で本番の機械学習インフラを設計し、デプロイした経験から、これには満足感があると証言できますが、飛行機のメンテナンスが飛行機の操縦と同じように、モデリングとは無縁のものなのです。

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

ウェブサイトに表示されている各オプションにはチュートリアルが含まれており、チュートリアルを完了すると、<>のような画面が表示されます。



You are now ready to run your first Jupyter notebook!

これで、最初の Jupyter ノートブックを実行する準備が整いました！

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.

ジャーゴン Jupyter Notebook (ジュピターノートブック)： フォーマットされたテキスト、コード、画像、ビデオなど、さまざまなものを持った1つのインターラクティブなドキュメントに含めることができるソフトウェアの一部です。Jupyterは、多くの学術分野や産業界で広く使用され、多大な影響を与えたことにより、ソフトウェアに対する最高の栄誉である ACM Software System Award を受賞しました。Jupyter Notebookは、データサイエンティストがディープラーニングモデルの開発や操作に最も広く使用されているソフトウェアです。

Running Your First Notebook

最初のノートブックを実行する

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).

ノートブックは、本書で紹介されているのと同じ順番になるように、章ごとに、そしてノートブック番号で表示されています。つまり、一番最初に表示されるノートブックが、今使うべきノートブックということになります。このノートブックを使って、犬や猫の写真を認識するモデルを学習させることになります。そのために、犬や猫の写真のデータセットをダウンロードし、それを使ってモデルを学習することになります。データセットとは、画像、メール、金融指標、音など、さまざまなデータの束のことです。モデルの学習に適したデータセットは、自由に利用できるものがたくさんあります。これらのデータセットの多くは、研究を進めるために学者が作成したもので、多くはコンペティション用に提供されています（データサイエンティストが最も正確なモデルを持つ者を競うコンペティションがあります！）また、他のプロセスの副産物（財務報告書など）であるものもあります。

note: Full and Stripped Notebooks: There are two folders containing different versions of the notebooks.

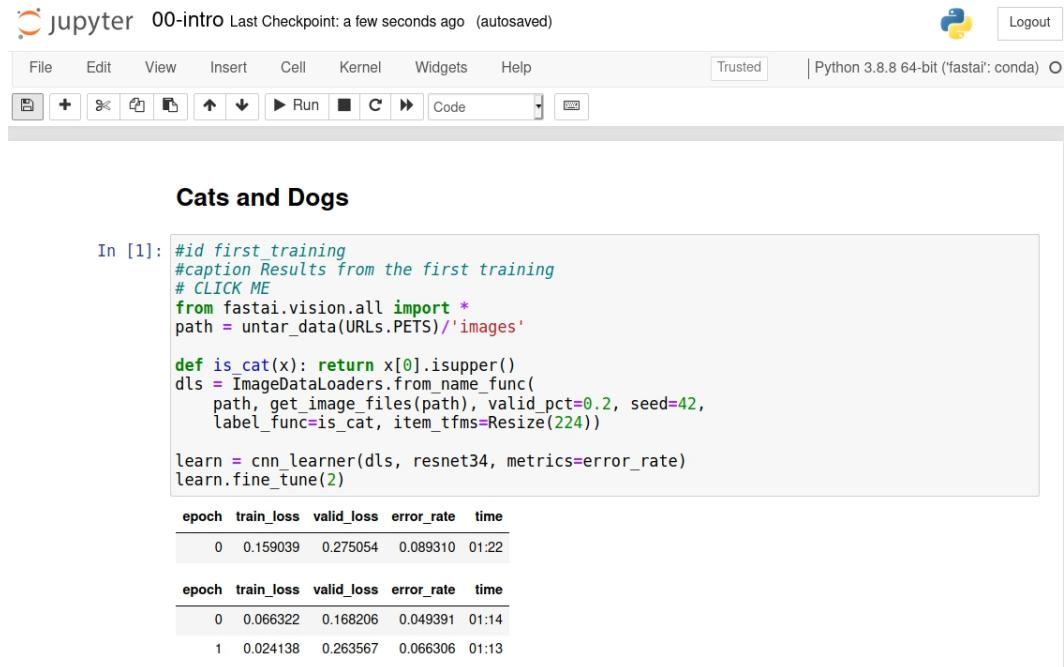
The *full* folder contains the exact notebooks used to create the book you're reading now, with all the prose and outputs.

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.

note: フルノートとストリップドノート： ノートブックには 2 つのフォルダがあり、異なるバージョンのノートブックが含まれています。完全版には、あなたが今読んでいる本を作るために使われたノートブックが、すべての散文と出力とともに収められています。ストリップ版には、見出しとコードセルは同じですが、出力とプロフィールがすべて削除されています。本の一部を読み終えたら、本を閉じたまま、剥がされたノートブックに目を通し、実行する前に各セルが何を示すのかがわかるかどうか試してみることをお勧めします。また、そのコードが何を示しているのかを思い出してみてください。

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).

ノートブックを開くには、そのノートブックをクリックするだけです。ノートブックが開き、<>のような表示になります（プラットフォームによって細部に若干の違いがある場合がありますが、その違いは無視してかまいません）。



The screenshot shows a Jupyter Notebook interface. At the top, there's a toolbar with File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3.8.8 64-bit ('fastai': conda), Logout, and other icons. Below the toolbar is a code editor cell containing Python code for training a model on 'Cats and Dogs' images. The code includes imports from fastai.vision.all, defines a function is_cat, creates data loaders dls, and initializes a learner learn. It then shows two tables of training metrics:

```
In [1]: #id first_training
#caption Results from the first training
# CLICK ME
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(2)
```

epoch	train_loss	valid_loss	error_rate	time
0	0.159039	0.275054	0.089310	01:22

epoch	train_loss	valid_loss	error_rate	time
0	0.066322	0.168206	0.049391	01:14
1	0.024138	0.263567	0.066306	01:13

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

ノートは細胞で構成されています。セルには、大きく分けて 2 つの種類があります：

- Cells containing formatted text, images, and so forth. These use a format called *markdown*, which you will learn about soon.
- フォーマットされたテキストや画像などを含むセル。これらはマークダウンと呼ばれるフォーマットを使用します。

- 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).

実行可能なコードを含むセルで、出力はすぐ下に表示されます（プレーンテキスト、テーブル、画像、アニメーション、サウンド、あるいは対話型アプリケーションなど）。

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 ノートブックは、編集モードとコマンドモードの 2 つのモードのいずれかになります。編集モードでは、キーボードでタイプすると、通常の方法で文字がセルに入力されます。しかし、コマンドモードでは、カーソルが点滅することなく、キーボードのキーはそれぞれ特別な機能を持つようになります。

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.

続行する前に、キーボードの Escape キーを押してコマンドモードに切り替えてください（すでにコマンドモードになっている場合は、何もしませんので、念のため今押してください）。すべての機能の完全なリストを見るには、H を押してください：Escape を押すと、このヘルプ画面が消えます。コマンドモードでは、ほとんどのプログラムと異なり、Control や Alt などを押し続ける必要がなく、必要な文字キーを押すだけでコマンドを実行できることに注意してください。

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.

C を押すと、セルのコピーを作成できます（セルの周囲に輪郭が表示され、セルが選択されている必要があります。）次に V キーを押すと、そのコピーが貼り付けられます。

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:

「CLICK ME」という行で始まるセルをクリックして選択します。その行の最初の文字は、その後に続くものが Python のコメントであることを示すので、セルを実行するときには無視されます。このセルの残りの部分は、信じられないかもしれません、猫と犬を認識するための最先端のモデルを作成し、訓練するための完全なシステムです。では、さっそく訓練してみましょう！そのためには、キーボードの Shift-Enter を押すか、ツールバーの再生ボタンを押すだけです。すると、次のようなことが起こるので、数分待ってください：

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.

Oxford-IIIT ペットデータセットと呼ばれる、37 種類の犬猫の画像 7,349 枚を含むデータセットが、fast.ai データセットコレクションからお使いの GPU サーバーにダウンロードされ、その後抽出されます。

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.

コンペティションで優勝したモデルを用いて 130 万枚の画像で学習済みのプレトレーニングモデルがインターネットからダウンロードされます。

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.

この学習済みモデルを、最新の転移学習で微調整し、犬や猫を認識するために特別にカスタマイズしたモデルを作成します。

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 (<>first_training>>):

最初の 2 つのステップは、GPU サーバー上で一度だけ実行すればよい。このセルを再度実行する場合は、データセットとモデルを再度ダウンロードするのではなく、すでにダウンロードされているものを使用します。それでは、セルの中身と結果 (<>) を見てみましょう：

```
In [ ]: #id first_training
#caption Results from the first training
# CLICK ME
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 = vision_learner(dls, resnet34, metrics=error_rate)
learn.fine_tune(1)
```

epoch	train_loss	valid_loss	error_rate	time
0	0.180385	0.023942	0.006766	00:16

epoch	train_loss	valid_loss	error_rate	time
0	0.056023	0.007580	0.004060	00:20

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.

おそらく、本に書かれているのと全く同じ結果は得られないでしょう。モデルのトレーニングには、小さなランダムな変動の原因がたくさん含まれています。しかし、この例では一般的に 0.02 を大きく下回るエラーレートを見ることができます。

important: Training 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: トレーニング時間: ネットワークの速度にもよりますが、訓練済みモデルとデータセットのダウンロードに数分かかる場合があります。

`fine_tune` を実行すると、1分ほどかかるかもしれません。本書のモデルも、あなた自身のモデルも、トレーニングに数分かかることが多いので、この時間を有効に使うための良いテクニックを思いつくとよいでしょう。例えば、モデルがトレーニングしている間、次のセクションを読み続けたり、別のノートを開いてコーディングの実験に使ったりするのです。

Sidebar: This Book Was Written in Jupyter Notebooks

サイドバー: 本書は Jupyter Notebooks で書かれました。

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.

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

本書は Jupyter ノートブックを使って書いたので、本書のほぼすべてのグラフ、表、計算について、自分で再現するために必要な正確なコードを紹介することにしています。そのため、本書では、コードのすぐ後に表や写真、あるいはテキストが表示されることが非常に多くなっています。この本のウェブサイトに行けば、すべてのコードを見る事ができますし、自分ですべての例を実行したり修正したりすることもできます。

この本の中で、表を出力するセルがどのように見えるかを見ていただきました。ここでは、テキストを出力するセルの例を示します:

In []:

1+1

Out[]:

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 は常に最後の行の結果を印刷または表示します（ある場合）。例えば、画像を出力するセルの例を以下に示します：

In []:

```
img = PILImage.create(image_cat0) img.to_thumb(192)
```

End sidebar

サイドバーを終了する

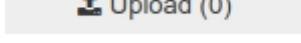
So, how do we know if this model is any good? In the last column of the table you can see the error rate, which is the proportion of images that were incorrectly identified. The error rate serves as our metric—our measure of model quality, chosen to be intuitive and comprehensible. As you can see, the 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!

では、このモデルが優れているかどうかは、どうすればわかるのでしょうか。表の最後の列には、誤認識率が表示されています。これは、誤認識された画像の割合を示しています。このエラーレートは、直感的で分かりやすいように選んだ、モデルの品質を測る指標となります。ご覧のように、学習時間はわずか数秒（データセットと学習済みモデルの1回分のダウンロードは含まず）であるにもかかわらず、モデルはほぼ完璧なものとなっています。この精度は、ほんの10年前に誰も達成できなかったものです！

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:

最後に、このモデルが実際に機能することを確認してみましょう。犬や猫の写真を撮ってきてください。手元にない場合は、Google Images で検索して、そこで見つけた画像をダウンロードしてください。では、アップローダーが定義されたセルを実行してみてください。クリックできるボタンが出力されるので、分類したい画像を選択することができます：

In []:

```
#hide_output uploader = widgets.FileUpload() uploader  
FileUpload(value={}, description='Upload')  

```

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:

これで、アップロードされたファイルをモデルに渡すことができます。線画や漫画などではなく、一匹の犬や猫の鮮明な写真であることを確認してください。ノートブックには、それが犬か猫か、どの程度の自信があるのかが書かれています。うまくいけば、モデルがいい仕事をしたことがわかるはずです：

In []:

```
#hide # For the book, we can't actually click an upload button, so we fake it uploader = SimpleNamespace(data = [images/chapter1_cat_example.jpg])
```

In []:

```
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.

Probability it's a cat: 1.000000

Congratulations on your first classifier!

これって、猫ですか？ 本當です。

猫である確率 1.000000

初めてのクラシファイヤー、おめでとうございます！

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.

でも、これってどういうことなんでしょう？ 実際に何をしたのでしょうか？ これを説明するためには、もう一度ズームアウトして全体像を把握しましょう。

What Is Machine Learning?

機械学習とは何か？

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.

あなたの分類器はディープラーニングモデルです。すでに述べたように、ディープラーニングモデルはニューラルネットワークを使用します。ニューラルネットワークはもともと1950年代のもので、最近の進歩により非常に強力になりました。

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.

もうひとつ重要なのは、ディープラーニングは機械学習という一般的な学問の中の最新の分野に過ぎないということです。分類モデルを学習させる際に行ったことの本質を理解するためには、ディープラーニングを理解する必要はないでしょう。自分のモデルとその学習過程が、機械学習全般に適用される概念の例であることが分かれば十分です。

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.

そこでこの項では、機械学習とは何かについて説明します。主要な概念を見て、それらを紹介した元のエッセイにどのように遡ることができるかを示していきます。

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.

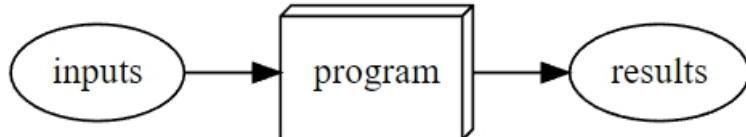
機械学習は、通常のプログラミングと同様に、コンピュータに特定のタスクを実行させるための方法です。しかし、前節で行った「写真に写っている犬と猫を見分ける」という作業を、通常のプログラミングで行うにはどうしたらいいのでしょうか。そのためには、コンピュータに必要な手順を正確に書き込まなければなりません。

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 <> (where *inputs* might be an unsorted list, and *results* a sorted list).

通常、プログラムを書くときに、タスクを完了させるための手順を書くのは簡単です。もし、その作業を手作業で行うとしたら、どのような手順で行うかを考え、それをコードに変換すればよいのです。例えば、リストをソートする関数を書くことができます。一般的には、<>のような関数を書きます（入力はソートされていないリストで、結果はソートされたリストです）。

In []:

```
#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!

写真に写っているものを認識するとき、私たちはどのような手順を踏んでいるのでしょうか。それは、私たちが意識することなく、すべて脳内で行われているため、実はわかっていないのです！

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:

コンピュータの黎明期、1949年にIBMの研究者アーサー・サミュエルが、コンピュータにタスクを実行させる別の方針を研究し始め、それを機械学習と呼びました。1962年に発表した古典的なエッセイ「人工知能（Artificial Intelligence）」の中で、こう書いています：

: 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.

：このような計算のためにコンピュータをプログラミングするのは、せいぜい困難な作業である。それは、コンピュータ自体に複雑さがあるからではなく、むしろ、そのプロセスの微細なステップを、最も気の遠くなるような細部に至るまで詳述しなければならないからである。コンピュータは、プログラマーなら誰でも知っているように、巨大な脳みそではなく、巨大なバカなのだ。

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):

彼の基本的な考え方とは、問題を解決するために必要な正確な手順をコンピュータに伝えるのではなく、解決すべき問題の例を示し、それを解決する方法をコンピュータ自身に考えさせるというものでした。1961年には、チェッカーで遊ぶプログラムが、コネティカット州のチャンピオンに勝つほど学習したのです！1961年、彼のチェッカープログラムは、コネチカット州のチャンピオンに勝つほど多くのことを学びました：

: 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.

：もし我々が、現在の重み付けの有効性を実際のパフォーマンスでテストする自動的な手段を用意し、パフォーマンスを最大にするように重み付けを変更するメカニズムを提供するとしよう。このような手順が完全に自動化できること、そしてそのようにプログラムされた機械がその経験から「学習」することは、詳細を説明するまでもないだろう。

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

この短い文章には、強力なコンセプトがいくつも埋め込まれている：

- The idea of a "weight assignment"
"重み付け"という考え方
- The fact that every weight assignment has some "actual performance"
すべての重量割り当てに、何らかの"実績"があるということ。
- The requirement that there be an "automatic means" of testing that performance,
その性能をテストする「自動的な手段」があることを要求しています、
- The need for a "mechanism" (i.e., another automatic process) for improving the performance by changing
the weight assignments
ウェイトの割り当てを変更することで性能を向上させる「仕組み」（＝別の自動処理）
が必要であること

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*.

これらの概念を1つずつ取り上げ、実際にどのように組み合わされるかを理解しましょう。
まず、サミュエルが言う「ウェイト割り当て」の意味を理解する必要があります。

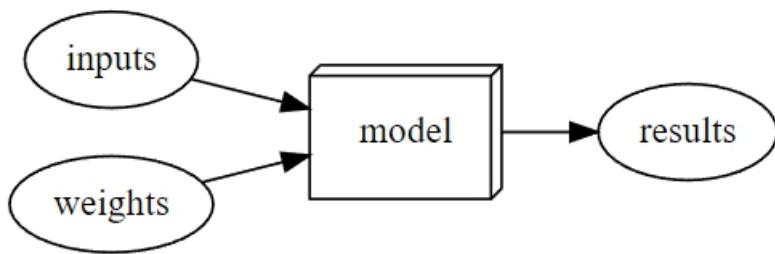
Weights are just variables, and a weight assignment is a particular choice of values for those variables. The program's inputs are values that it processes in order to produce its results—for instance, taking image pixels as inputs, and returning the classification "dog" as a result. The program's weight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

ウェイトとは単なる変数であり、ウェイトの割り当てとは、その変数の値の特定の選択である。例えば、画像のピクセルを入力とし、「犬」という分類を結果で返す。プログラムの重み付けは、プログラムがどのように動作するかを定義するその他の値です。

Since they will affect the program they are in a sense another kind of input, so we will update our basic picture in <> and replace it with <> in order to take this into account.
プログラムに影響を与えるので、ある意味、別の種類の入力である。これを考慮するために、<>の基本図を更新して<>に置き換える。

In []:

```
#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.

私たちは、ボックスの名前をプログラムからモデルに変えました。これは、現代の用語に従うためであり、モデルが特別な種類のプログラムであることを反映するためです：それは、ウェイトに応じて、多くの異なることができるものです。重さによってさまざまなことができるプログラムなのです。例えば、サミュエルのチェッカープログラムでは、重みの値が違えば、チェッカーをプレイする戦略も違ってきます。

(By the way, what Samuel called "weights" are most generally referred 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.)

(ちなみに、サミュエルが「重み」と呼んだものは、最近では一般的にモデルパラメータと呼ばれることが多いようです。ウェイトという言葉は、ある種のモデルパラメータにのみ使われる言葉である)。

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.

次に、サミュエルは、現在のウェイト割り当ての有効性を実際のパフォーマンスでテストする自動的な手段が必要であると言った。彼のチェッカーズ・プログラムの場合、モデルの「実際の性能」とは、どれだけうまくプレイできるかということだ。そして、2つのモデルを互いに対戦させ、どちらが勝つかを見ることで、その性能を自動的にテストすることができるのです。

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.

最後に、性能を最大化するために、ウェイトの割り当てを変更するメカニズムが必要だと言います。例えば、勝ったモデルと負けたモデルの重みの差を見て、勝つ方向に少し重みを調整することができます。

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.

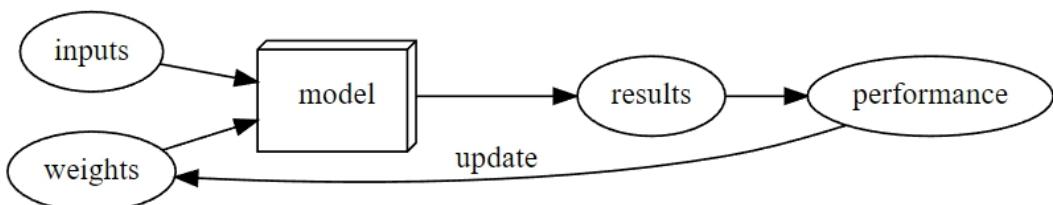
このような手順を完全に自動化することができ、そのようにプログラムされた機械は、その経験から「学習」すると述べた理由がわかります。つまり、手動で重みを調整してモデルを改良するのではなく、性能に基づいて調整を行う自動化されたメカニズムに頼ればいいのです。

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

<>は、サミュエルが考えた機械学習モデルのトレーニングの全体像を示している。

In []:

```
#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).

モデルの結果(例えば、チェックergameの手)と性能(例えば、ゲームに勝つか、どれくらい早く勝つか)は区別されることに注意してください。

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.

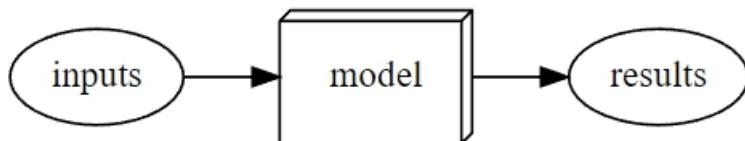
また、モデルの学習が完了すると、つまり最終的な、最良の、お気に入りの重みの割り当てを選択すると、重みはもう変化しないので、重みはモデルの一部であると考えることができます。

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

したがって、学習後のモデルを実際に使用する場合は、<<using_model>>のようになります。

In []:

```
#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 <<basic_program>>, 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*.

この図は、<>の元の図と同じですが、*program* という単語が *model* に置き換わっています。これは重要な洞察です。訓練されたモデルは、通常のコンピュータプログラムと同じように扱うことができるのです。

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.

専門用語です： 機械学習： 専門用語： 機械学習： 個々のステップを手作業でコーディングするのではなく、コンピュータに経験から学習させることによって開発されるプログラムのトレーニングのこと。

What Is a Neural Network?

ニューラルネットワークとは？

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.

チェックカーブログラムのモデルを想像するのは、それほど難しいことではありません。チェックカーズにはさまざまな戦略がコード化されており、検索メカニズムもある。そして、戦略の選択方法、検索時に盤面のどの部分に注目するかなどを重みで変えることができる。しかし、画像認識プログラム、テキスト理解、その他多くの興味深い問題に対して、どの

ようなモデルが考えられるかは、まったく明らかではありません。

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.

そこで、重みを変えるだけで、どんな問題にも対応できるような、柔軟な関数が必要になる。驚くべきことに、このような関数は実際に存在するのです！それは、すでに説明したニューラルネットワークです。つまり、ニューラルネットワークを数学的な関数としてとらえると、その重みによって極めて柔軟に変化する関数であることがわかる。普遍的近似定理と呼ばれる数学的証明によれば、この関数は理論的にはどんな問題でもどんな精度でも解けることがわかります。ニューラルネットワークが非常に柔軟であるということは、実際には、ニューラルネットワークが適切なモデルであることが多く、ニューラルネットワークを学習させるプロセス、つまり、良い重みの割り当てを見つけることに力を注ぐことができる、ということです。

But what about that process? One could imagine that you might need to find a new "mechanism" for automatically updating weights 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!

しかし、そのプロセスはどうでしょうか。問題ごとに自動的に重みを更新する新しい「メカニズム」を見つける必要があるかもしれませんと想像することができます。これでは手間がかかる。そこで、ニューラルネットワークの重みを更新して、どのような課題にも対応できるようにするための、完全に一般的な方法が必要です。便利なことに、これも存在します！

This is called *stochastic gradient descent* (SGD). We'll see how neural networks and SGD work in detail in <>chapter_mnist_basics>, 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.*

これは確率的勾配降下法 (SGD) と呼ばれるものです。ニューラルネットワークと SGD がどのように機能するかは、<>で詳しく説明しますし、万能近似定理も説明します。しかし、今はその代わりに、サミュエル自身の言葉を使うことにしよう： このような手順が完全に自動化できること、そしてそのようにプログラムされた機械がその経験から「学習」することを確認するために、その詳細を説明する必要はないだろう。

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?"

J: SGD もニューラルネットも、数学的に複雑なものではありませんから、ご心配なく。どちらもほぼ完全に足し算と掛け算だけで仕事をします(でも、足し算と掛け算はたくさんしますよ！)。詳細を見た学生たちの主な反応はこうです： "これで終わりなの？"ということです。

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.

つまり、要約すると、ニューラルネットワークは特殊な機械学習モデルであり、サミュエルの当初の構想にぴったりと当てはまる。ニューラルネットワークが特別なのは、柔軟性が高いからです。つまり、正しい重みを見つけるだけで、非常に幅広い問題を解決できるのです。確率的勾配降下法では、その重みの値を自動的に求めることができるので、これは強力です。

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

ズームアウトした後、ズームインして、Samuel のフレームワークを使って、画像分類の問題をもう一度考えてみましょう。

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."

入力は画像である。重みとはニューラルネットの重みのことです。モデルとはニューラルネットのことです。結果はニューラルネットが計算した値です。"犬"とか "猫"とか。

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.

次に、現在の重みの割り当ての有効性を、実際のパフォーマンスという観点から自動的にテストする手段についてはどうでしょうか。実際の性能」を決めるのは簡単です。モデルの性能を「正解を予測する精度」と定義すればいいのです。

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.

このように、SGD が重み付けを更新するメカニズムであると仮定すると、画像分類器は Samuel が思い描いたような機械学習モデルであることがわかるでしょう。

A Bit of Deep Learning Jargon

ディープラーニングの専門用語の一例

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:

サミュエルは 1960 年代に活動していましたが、それ以来、用語は変化しています。ここでは、これまで説明したすべてのピースについて、現代のディープラーニング用語を紹介します：

- 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).

モデルの関数形はアーキテクチャと呼ばれます(ただし、アーキテクチャの同義語としてモデルを使用することがあるため、混乱することがあります)。

- The *weights* are called *parameters*.

重みはパラメータと呼ばれます。

- The *predictions* are calculated from the *independent variable*, which is the *data* not including the *labels*.
予測値は、ラベルを含まないデータである独立変数から計算されます。

- The *results* of the model are called *predictions*.

モデルの結果は予測値と呼ばれます。

- The measure of *performance* is called the *loss*.

性能の指標は損失と呼ばれます。

- 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."

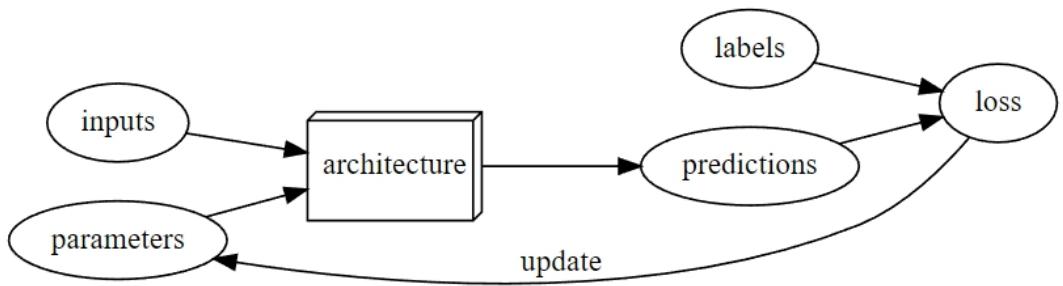
損失は予測値だけでなく、正しいラベル(ターゲットまたは従属変数とも呼ばれる)にも依存する。例えば、"dog" や "cat" などである。

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

これらの変更を行った後、<<training_loop>>の図は、<<detailed_loop>>のようになります。

In []:

```
#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

機械学習に内在する限界

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

この図から、ディープラーニングモデルのトレーニングに関する基本的なことが見えてきました:

- A model cannot be created without data.
モデルはデータがなければ作れません。
- A model can only learn to operate on the patterns seen in the input data used to train it.
モデルはデータなしで作成することはできません。モデルは、それを訓練するために使用される入力データに見られるパターンに基づいてのみ操作することを学習することができます。
- This learning approach only creates *predictions*, not recommended *actions*.
この学習アプローチは、予測を作成するだけで、推奨される行動を作成するわけではありません。
- 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).
入力データの例だけでは不十分で、そのデータに対するラベルも必要です（例えば、犬と猫の写真だけではモデルの学習には不十分で、どれが犬で、どれが猫かというラベルが必要です）。

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.

一般的に、「データが足りない」と言う組織の多くは、実は「ラベル付きデータが足りない」ことを意味していることが分かっています。もし、ある組織がモデルを使って実際に何かをすることに興味があるのなら、おそらく彼らはモデルを実行する予定の入力があるはずです。そして、おそらくしばらくは他の方法で（例えば、手動で、あるいはヒューリスティックなプログラムで）それを行っているので、そのプロセスから得られたデータを持っているはずである！例えば、放射線科の診療所は、ほぼ間違いなく医療用スキャンのアーカイブを持っています（患者が時間とともにどのように進行しているかを確認できる必要があるため）。しかし、これらのスキャンには、診断や介入のリストを含む構造化ラベルがないかもしれません（放射線科医は通常、構造化データではなく、フリーテキストの自然言語レポートを作成するから）。この本では、ラベリングアプローチについて何度も説明します。なぜなら、ラベリングは実務上非常に重要な問題だからです。

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.

この種の機械学習モデルは予測しかできないため（つまり、ラベルの再現を試みる）、組織の目標とモデルの能力の間に大きなギャップが生じる可能性があります。例えば、本書では、ユーザーが購入する可能性のある商品を予測するレコメンデーションシステムの作り方を学ぶことができます。これは E コマースでよく使われるもので、例えばホームページに表示される商品をカスタマイズして、ランキング上位の商品を表示するようなものです。しかし、このようなモデルは、一般に、ユーザーとその購入履歴（インプット）と、そのユーザーが購入したり見たりしたもの（ラベル）を見て作られます。つまり、そのモデルは、ユーザーが最も興味を持ちそうな新しい商品ではなく、すでに持っていたり、知っている商品について教えてくれる可能性が高いということです。これは、例えば地元の書店の専門家が、質問をしてユーザーの好みを把握した上で、聞いたこともないような作家やシリーズを紹介するのとは大きく異なる。

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

もう一つの重要な洞察は、モデルが環境とどのように相互作用するかを考えることから得られます。これは、次のようなフィードバックループを生み出す可能性があります：

- 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.

過去に逮捕された場所に基づいて、予測的な取り締まりモデルが作成される。実際には、犯罪の予測ではなく、検挙数の予測であるため、既存の取り締まりプロセスのバイアスを反映しているに過ぎない部分もあります。

- Law enforcement officers then might use that model to decide where to focus their police activity, resulting in increased arrests in those areas.

警察官は、そのモデルを使って、どこに警察活動を集中させるかを決め、その結果、その地域での検挙件数が増えるかもしれません。

- Data on these additional arrests would then be fed back in to retrain future versions of the model.

その結果、その地域の検挙件数が増加する。この検挙件数のデータは、将来のモデルの再トレーニングにフィードバックされる。

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.

これは正のフィードバックループであり、モデルを使えば使うほど、データの偏りが増し、さらにモデルの偏りが増すという具合に、繰り返される。

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 <>.

フィードバックループは、商業的な場面でも問題を引き起こすことがあります。例えば、動画推薦システムが、動画をよく見る人（例えば、陰謀論者や過激派は、平均よりも多くのオンライン動画コンテンツを見る傾向がある）が消費するコンテンツを推薦するように偏ってしまい、その結果、それらのユーザーが動画の消費を増やして、その種類の動画がより多く推薦されることになるかもしれません。この話題については、<>で詳しく検討します。

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.

さて、理論のベースが見えたところで、コードの例に戻って、コードが先ほど説明したプロセスにどう対応しているかを詳しく見てみましょう。

How Our Image Recognizer Works

画像認識装置の仕組み

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).

では、画像認識エンジンのコードが、これらの考え方に対応しているかを見て

みましょう。それぞれの行を別のセルに入れ、それぞれの行が何をしているのかを見てみましょう（すべてのパラメータの詳細についてはまだ説明しませんが、重要な部分については説明します； 完全な詳細についてはこの本の後半で説明します）。

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

最初の行は、fastai.vision ライブラリのすべてをインポートしています。

```
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.

これによって、さまざまなコンピュータビジョンモデルを作成するために必要な関数やクラスがすべて揃いました。

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.

J: Python のコーダーの多くは、このようにライブラリ全体をインポートする (`import *`構文を使う) のは避けるように勧めています。しかし、Jupyter ノートブックのようなインタラクティブな作業では、この方法はとても有効です。fastai ライブラリは、このようなインタラクティブな使い方をサポートするために特別に設計されており、必要な部分のみをあなたの環境にインポートしてくれるのです。

The second line downloads a standard dataset from the [fast.ai datasets collection](#) (if not previously downloaded) to your server, extracts it (if not previously extracted), and returns a Path object with the extracted location:

2 行目は、fast.ai datasets コレクションから標準データセットをサーバーにダウンロードし（以前にダウンロードされていない場合）、それを抽出し（以前に抽出されていない場合）、抽出した場所を Path オブジェクトで返します：

```
path = untar_data(URLs.PETS)/'images'  
パス = untar_data(URLs.PETS)/'images'
```

S: Throughout my time studying at fast.ai, and even still today, I've learned a lot about productive coding practices. The fastai library and fast.ai notebooks are full of great little tips that have helped make me a better programmer. For instance, notice that the fastai library doesn't just return a string containing the path to the dataset, but a `Path` object. This is a really useful class from the Python 3 standard library that makes accessing files and directories much easier. If you haven't come across it before, be sure to check out its documentation or a tutorial and try it out. Note that the <https://book.fast.ai%5Bwebsite>] contains links to recommended tutorials for each chapter. I'll keep letting you know about little coding tips I've found useful as we come across them.

S: fast.ai で勉強している間、そして今でも、生産的なコーディングの実践について多くのことを学んでいます。fastai ライブラリや fast.ai のノートブックには、私をより良いプログラマーにするための素晴らしい小さなヒントがたくさん詰まっています。例えば、fastai ライブラリは、データセットへのパスを含む文字列を返すだけでなく、`Path` オブジェクトを返すことについて注目してください。これは Python 3 標準ライブラリにあるとても便利なクラスで、ファイルやディレクトリへのアクセスをより簡単にすることができます。もしあなたがまだ使ったことがないのであれば、ぜひドキュメントやチュートリアルを読んで試してみてください。なお、【<https://book.fast.ai%5Bwebsite>】には、各章のおすすめチュートリアルへのリンクが掲載されています。今後も、便利だと思った小さなコーディングのヒントを見つけたら、お知らせしたいと思います。

In the third line we define a function, `is_cat`, which labels cats based on a filename rule provided by the dataset creators:

3 行目では、データセット作成者が提供したファイル名ルールに基づいて猫のラベルを付ける関数、`is_cat` を定義しています：

```
def is_cat(x): return x[0].isupper()
```

We use that function in the fourth line, which tells fastai what kind of dataset we have and how it is structured:

その関数を 4 行目で使って、fastai にデータセットの種類と構造を伝えています：

```
dls = ImageDataLoaders.from_name_func(  
    path, get_image_files(path), valid_pct=0.2, seed=42,  
    label_func=is_cat, item_tfms=Resize(224))
```

There are various different classes for different kinds of deep learning datasets and problems—here we're using `ImageDataLoaders`. The first part of the class name will generally be the type of data you have, such as `image`, or `text`.

深層学習のデータセットや問題の種類によって様々なクラスがあり、ここでは `ImageDataLoaders` を使用します。クラス名の最初の部分は、一般的に、画像やテキストなど、データの種類を表します。

The other important piece of information that we have to tell fastai is how to get the labels from the dataset. Computer vision datasets are normally structured in such a way that the label for an image is part of the filename, or path—most commonly the parent folder name. fastai comes with a number of standardized labeling methods, and ways to write your own. Here we're telling fastai to use the `is_cat` function we just defined.

fastai に伝えるべきもう一つの重要な情報は、データセットからラベルを取得する方法です。コンピュータビジョンのデータセットは通常、画像のラベルがファイル名やパス（最も一般的なのは親フォルダ名）の一部になるような構造になっています。fastai には、標準的なラベル付け方法と、独自のラベル付け方法が用意されています。ここでは、先ほど定義した `is_cat` 関数を使用するように fastai に指示しています。

Finally, we define the Transforms that we need. A Transform contains code that is applied automatically during training; fastai includes many predefined Transforms, and adding new ones is as simple as creating a Python function. There are two kinds: `item_tfms` are applied to each item (in this case, each item is resized to a 224-pixel square), while `batch_tfms` are applied to a *batch* of items at a time using the GPU, so they're particularly fast (we'll see many examples of these throughout this book).

最後に、必要な Transforms を定義します。Fastai には多くの定義済みの Transform が含まれており、新しい Transform を追加するには Python 関数を作成するのと同じくらい簡単です。`item_tfms` は各アイテムに適用され（この場合、各アイテムは 224 ピクセル四方にリサイズされる）、`batch_tfms` は GPU を使って一度にアイテムのバッチに適用されるので特に高速です（この本ではこの例をたくさん見ていきますね）。

Why 224 pixels? This is the standard size for historical reasons (old pretrained models require this size exactly), but you can pass pretty much anything. If you increase the size, you'll often get a model with better results (since it will be able to focus on more details), but at the price of speed and memory consumption: the opposite is true if you decrease the size.

なぜ 224 ピクセルなのか？これは歴史的な理由で標準的なサイズです（古い事前学習モデルはこのサイズを正確に要求します）が、かなり多くのものを渡すことができます。サイズを大きくすると、（より詳細に焦点を当てる事ができるため）より良い結果を得ることができますですが、速度とメモリ消費の代償として、その逆となります。

Note: Classification and
Regression: *classification* and *regression* have very
specific meanings in machine learning. These are the two main

types of model that we will be investigating in this book. A classification model is one which attempts to predict a class, or category. That is, it's predicting from a number of discrete possibilities, such as "dog" or "cat." A regression model is one which attempts to predict one or more numeric quantities, such as a temperature or a location. Sometimes people use the word *regression* to refer to a particular kind of model called a *linear regression model*; this is a bad practice, and we won't be using that terminology in this book!

注：分類と回帰：分類と回帰は、機械学習において非常に特殊な意味を持っています。本書で調査するのは、主にこの2種類のモデルである。分類モデルとは、クラス（分類）を予測しようとするものです。つまり、"犬"や"猫"のような、いくつかの離散的な可能性から予測しようとするものです。回帰モデルは、温度や場所など、1つ以上の数値量を予測しようとするものです。回帰という言葉を、線形回帰モデルと呼ばれる特定の種類のモデルを指すために使う人が時々いますが、これは悪い習慣なので、この本ではこの用語を使いません！

The Pet dataset contains 7,390 pictures of dogs and cats, consisting of 37 different breeds. Each image is labeled using its filename: for instance the file *great_pyrenees_173.jpg* is the 173rd example of an image of a Great Pyrenees breed dog in the dataset. The filenames start with an uppercase letter if the image is a cat, and a lowercase letter otherwise. We have to tell fastai how to get labels from the filenames, which we do by calling `from_name_func` (which means that labels can be extracted using a function applied to the filename), and passing `is_cat`, which returns `x[0].isupper()`, which evaluates to True if the first letter is uppercase (i.e., it's a cat).

Pet データセットには、37種類の犬種からなる犬と猫の写真が 7,390 枚含まれています。例えば、*great_pyrenees_173.jpg* というファイルは、データセットに含まれるグレート・ピレニーズ種の犬の画像の 173 番目の例である、というように、各画像はファイル名を使ってラベル付けされている。ファイル名は、画像が猫の場合は大文字で始まり、そうでない場合は小文字で始まります。fastai には、ファイル名からラベルを取得する方法を伝えなければなりません。`from_name_func` (ファイル名に適用する関数を使ってラベルを抽出できることを意味します) を呼び出し、`is_cat` を渡します。この関数では、`x[0].isupper` (最初の文字が大文字 (つまり、猫) の場合は True と評価される) を返します。

The most important parameter to mention here is `valid_pct=0.2`. This tells fastai to hold out 20% of the data and *not use it for training the model at all*. This 20% of the data is called the *validation set*; the remaining 80% is called the *training set*. The validation set is used to measure the accuracy of the model. By default, the 20% that is held out is selected randomly. The parameter `seed=42` sets the *random seed* to the same value every time we run this code, which means we get the same validation set every time we run it—this way, if we change our model and retrain it, we know that any differences are due to the changes to the model, not due to having a different random validation set.

ここで最も重要なパラメータは `valid_pct=0.2` です。これは `fastai` に 20% のデータを保留し、モデルのトレーニングに一切使用しないように指示するものです。この 20% のデータを検証セットと呼び、残りの 80% をトレーニングセットと呼びます。検証セットはモデルの精度を測定するために使用されます。デフォルトでは、保留される 20% はランダムに選択されます。`seed=42` というパラメータは、このコードを実行するたびにランダムシードを同じ値に設定します。つまり、このコードを実行するたびに同じ検証セットを得ることができます。このようにすれば、モデルを変更して再トレーニングしても、その違いはモデルの変更によるものであり、ランダム検証セットの違いによるものではないことがわかります。

`fastai` will *always* show you your model's accuracy using *only* the validation set, *never* the training set. This is absolutely critical, because if you train a large enough model for a long enough time, it will eventually memorize the label of every item in your dataset! The result will not actually be a useful model, because what we care about is how well our model works on *previously unseen images*. That is always our goal when creating a model: for it to be useful on data that the model only sees in the future, after it has been trained.

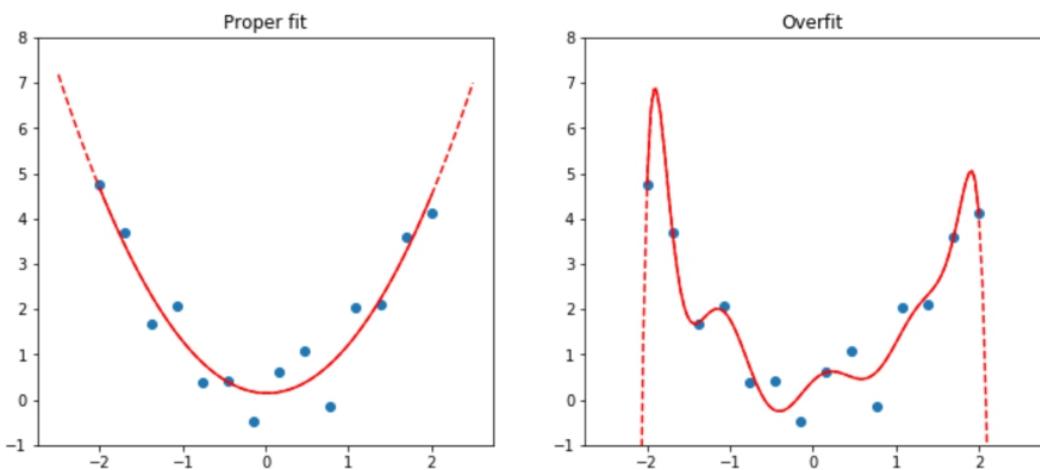
`fastai` は、常に検証セットのみを使用したモデルの精度を表示し、トレーニングセットを表示することはありません。これは非常に重要なことです。なぜなら、十分な大きさのモデルを十分な期間訓練すると、最終的にデータセット内のすべてのアイテムのラベルを記憶してしまうからです！その結果、有用なモデルとは言えなくなってしまいます。モデルを作るときの目標は常に、「モデルが学習した後に初めて見るデータに対して有用であること」なのです。

Even when your model has not fully memorized all your data, earlier on in training it may have memorized certain parts of it. As a result, the longer you train for, the better your accuracy will get on the training set; the validation set accuracy will also improve for a while, but eventually it will start getting worse as the model starts to memorize the training set, rather than finding generalizable underlying patterns in the data. When this happens, we say that the model is *overfitting*.

モデルがすべてのデータを完全に記憶していないくとも、トレーニングの初期には、ある部分を記憶していることがあります。検証セットの精度もしばらくは向上しますが、やがて、モデルがデータから一般化できる基本的なパターンを見つけるのではなく、トレーニングセットを記憶し始めるため、精度が低下していきます。このような場合、モデルはオーバーフィッティングしていると言えます。

<<img_overfit>> shows what happens when you overfit, using a simplified example where we have just one parameter, and some randomly generated data based on the function x^{**2} . As you can see, although the predictions in the overfit model are accurate for data near the observed data points, they are way off when outside of that range.

<>は、パラメータが 1 つだけで、関数 x^{**2} に基づきランダムに生成されたデータを使った単純化された例で、オーバーフィットが起こったときの様子を示しています。ご覧のように、オーバーフィットモデルの予測は、観測されたデータポイントの近くでは正確ですが、その範囲外では大きく外れています。



Overfitting is the single most important and challenging issue when training for all machine learning practitioners, and all algorithms. As you will see, it is very easy to create a model that does a great job at making predictions on the exact data it has been trained on, but it is much harder to make accurate predictions on data the model has never seen before. And of course, this is the data that will actually matter in practice. For instance, if you create a handwritten digit classifier (as we will very soon!) and use it to recognize numbers written on checks, then you are never going to see any of the numbers that the model was trained on—checks will have slightly different variations of writing to deal with. You will learn many methods to avoid overfitting in this book. However, you should only use those methods after you have confirmed that overfitting is actually occurring (i.e., you have actually observed the validation accuracy getting worse during training). We often see practitioners using overfitting avoidance techniques even when they have enough data that they didn't need to do so, ending up with a model that may be less accurate than what they could have achieved.

オーバーフィッティングは、すべての機械学習の実践者、そしてすべてのアルゴリズムにとって、学習時に最も重要で困難な問題です。お分かりのように、学習させたデータに対して優れた予測を行うモデルを作るのは非常に簡単ですが、そのモデルが見たこともないデータに対して正確な予測を行うのは、はるかに困難なことです。もちろん、これは実際に問題になるデータです。例えば、手書きの数字分類器を作成し（もうすぐ作成します！）、それを使って小切手に書かれた数字を認識する場合、モデルが学習した数字を見ることはありません-小切手には微妙に異なる書き方がありますから、対応する必要があります。本書では、オーバーフィッティングを回避するための方法を数多く学びます。しかし、これらの方法は、オーバーフィッティングが実際に起こっていることを確認した後（つまり、トレーニング中に検証精度が悪くなっていることを実際に観察した後）にのみ使用する必要があります。その必要がないほど十分なデータがあるにもかかわらず、オーバーフィッティング回避のテクニックを使ってしまい、結局、達成できたものよりも精度が低いモデルになってしまふという実務家がよく見受けられる。

important: Validation Set: When you train a model, you must *always* have both a training set and a validation set,

and must measure the accuracy of your model only on the validation set. If you train for too long, with not enough data, you will see the accuracy of your model start to get worse; this is called *overfitting*. fastai defaults `valid_pct` to 0.2, so even if you forget, fastai will create a validation set for you!

重要: バリデーション・セット モデルをトレーニングする際には、必ずトレーニングセットと検証セットの両方を用意し、検証セットでのみモデルの精度を測定する必要があります。fastai のデフォルトは `valid_pct` が 0.2 なので、忘れても fastai が検証セットを作ってくれます！

The fifth line of the code training our image recognizer tells fastai to create a *convolutional neural network* (CNN) and specifies what *architecture* to use (i.e. what kind of model to create), what data we want to train it on, and what *metric* to use:

画像認識器をトレーニングするコードの 5 行目は、fastai に畳み込みニューラルネットワーク (CNN) を作成するよう指示し、どのアーキテクチャを使うか（つまり、どのようなモデルを作成するか）、どのデータでそれをトレーニングするか、どのメトリックを使用するかを指定します：

```
learn = vision_learner(dls, resnet34, metrics=error_rate)
```

Why a CNN? It's the current state-of-the-art approach to creating computer vision models. We'll be learning all about how CNNs work in this book. Their structure is inspired by how the human vision system works.

なぜ CNN なのか？ それは、コンピュータビジョンモデルを作成するための現在の最先端アプローチだからです。本書では、CNN がどのように機能するのか、そのすべてを学ぶことができます。その構造は、人間の視覚システムの仕組みにヒントを得ています。

There are many different architectures in fastai, which we will introduce in this book (as well as discussing how to create your own). Most of the time, however, picking an architecture isn't a very important part of the deep learning process. It's something that academics love to talk about, but in practice it is unlikely to be something you need to spend much time on. There are some standard architectures that work most of the time, and in this case we're using one called *ResNet* that we'll be talking a lot about during the book; it is both fast and accurate for many datasets and problems. The 34 in `resnet34` refers to the number of layers in this variant of the architecture (other options are 18, 50, 101, and 152). Models using architectures with more layers take longer to train, and are more prone to overfitting (i.e. you can't train them for as many epochs before the accuracy on the validation set starts getting worse). On the other hand, when using more data, they can be quite a bit more accurate.

fastai には様々なアーキテクチャがあり、本書で紹介します（自分で作成する方法についても説明します）。しかし、ほとんどの場合、アーキテクチャを選ぶことは、深層学習のプロセスにおいてあまり重要なことではありません。学者が好んで話すことではありますが、実際にはあまり時間をかける必要はないでしょう。今回は ResNet と呼ばれるものを使っていますが、これはこの本の中でたくさんお話しします。`resnet34` の 34 は、このアーキテク

チャのバリエーションにおける層の数を意味します（他のオプションは 18、50、101、152 です）。層数の多いアーキテクチャを使用したモデルは、学習に時間がかかり、オーバーフィッティングを起こしやすくなります（検証セットでの精度が悪くなる前に、多くのエポック数を学習することができないなど）。一方、より多くのデータを使用することで、かなり精度が向上する可能性があります。

What is a metric? A *metric* is a function that measures the quality of the model's predictions using the validation set, and will be printed at the end of each *epoch*. In this case, we're using `error_rate`, which is a function provided by fastai that does just what it says: tells you what percentage of images in the validation set are being classified incorrectly. Another common metric for classification is `accuracy` (which is just $1.0 - \text{error_rate}$). fastai provides many more, which will be discussed throughout this book.

メトリックとは何ですか？ メトリックとは、検証セットを使ったモデルの予測の品質を測定する関数で、各エポックの終了時に出力されます。この場合、`error_rate` を使います。これは fastai が提供する関数で、その名の通り、検証セット内の画像の何パーセントが誤って分類されたかを教えてくれます。分類のもう一つの一般的な指標は `accuracy` ($1.0 - \text{error_rate}$) で、fastai には他にも多くの機能があり、この本で説明します。

The concept of a metric may remind you of *loss*, but there is an important distinction. The entire purpose of loss is to define a "measure of performance" that the training system can use to update weights automatically. In other words, a good choice for loss is a choice that is easy for stochastic gradient descent to use. But a metric is defined for human consumption, so a good metric is one that is easy for you to understand, and that hews as closely as possible to what you want the model to do. At times, you might decide that the loss function is a suitable metric, but that is not necessarily the case.

メトリックの概念は、損失を思い起こさせるかもしれません、重要な区別があります。損失の全目的は、学習システムが自動的に重みを更新するために使用できる「性能の尺度」を定義することです。言い換えれば、ロスの良い選択とは、確率的勾配降下法にとって使いやすい選択である。しかし、指標は人間が消費するために定義されるものですから、良い指標とは、自分にとって理解しやすく、モデルに何をさせたいかにできるだけ忠実なものです。時には、損失関数が適切なメトリックであると判断することもあるかもしれません、必ずしもそうではありません。

`vision_learner` also has a parameter `pretrained`, which defaults to `True` (so it's used in this case, even though we haven't specified it), which sets the weights in your model to values that have already been trained by experts to recognize a thousand different categories across 1.3 million photos (using the famous [ImageNet dataset](#)). A model that has weights that have already been trained on some other dataset is called a *pretrained model*. You should nearly always use a pretrained model, because it means that your model, before you've even shown it any of your data, is already very capable. And, as you'll see, in a deep learning model many of these capabilities are things you'll need, almost regardless of the details of your project. For instance, parts of pretrained models will handle edge, gradient, and color detection, which are needed for many tasks.

`vision_learner` には `pretrained` というパラメータがあり、デフォルトは `True` です（この場合、指定していないにもかかわらず、このパラメータが使われます）。他のデータセット

すでに訓練された重みを持つモデルは、事前訓練済みモデルと呼ばれます。これは、データを見せる前のモデルが、すでに非常に高性能であることを意味します。そして、これからわかるように、ディープラーニングモデルでは、これらの機能の多くが、プロジェクトの詳細とは無関係に必要とされるものです。例えば、事前学習済みモデルの一部は、多くのタスクで必要とされるエッジ、グラデーション、および色の検出を処理します。

When using a pretrained model, `vision_learner` will remove the last layer, since that is always specifically customized to the original training task (i.e. ImageNet dataset classification), and replace it with one or more new layers with randomized weights, of an appropriate size for the dataset you are working with. This last part of the model is known as the *head*.

学習済みモデルを使用する場合、`vision_learner` は最後のレイヤーを削除します。なぜなら、最後のレイヤーは常に元の学習タスク (ImageNet データセットの分類など) に合わせて特別にカスタマイズされており、作業中のデータセットに適したサイズの、重みがランダムな 1 以上の新しいレイヤーと置き換えます。このモデルの最後の部分は、ヘッドと呼ばれます。

Using pretrained models is the *most* important method we have to allow us to train more accurate models, more quickly, with less data, and less time and money. You might think that would mean that using pretrained models would be the most studied area in academic deep learning... but you'd be very, very wrong! The importance of pretrained models is generally not recognized or discussed in most courses, books, or software library features, and is rarely considered in academic papers. As we write this at the start of 2020, things are just starting to change, but it's likely to take a while. So be careful: most people you speak to will probably greatly underestimate what you can do in deep learning with few resources, because they probably won't deeply understand how to use pretrained models.

事前学習済みモデルの使用は、より正確なモデルを、より早く、より少ないデータで、より少ない時間と費用で学習させるための最も重要な方法です。ということは、事前学習済みモデルの使用は、学術的な深層学習において最も研究されている分野であると思うかもしれません...しかし、それは非常に大きな間違いです! プレトレーニングモデルの重要性は、一般的にほとんどのコース、書籍、ソフトウェアライブラリの機能では認識されておらず、議論されていませんし、学術論文でもほとんど考慮されていません。これを書いている 2020 年初頭の時点では、状況は変わり始めているところですが、しばらく時間がかかりそうです。ですから、注意してください: あなたが話すほとんどの人は、少ないリソースで深層学習でできることをおそらく大幅に過小評価するでしょう、なぜなら彼らはおそらくプレトレーニングされたモデルの使い方を深く理解していないから。

Using a pretrained model for a task different to what it was originally trained for is known as *transfer learning*. Unfortunately, because transfer learning is so under-studied, few domains have pretrained models available. For instance, there are currently few pretrained models available in medicine, making transfer learning challenging to use in that domain. In addition, it is not yet well understood how to use transfer learning for tasks such as time series analysis.

プリトレーニングされたモデルを、元々トレーニングされたものとは異なるタスクに使うことは、転移学習として知られています。残念ながら、転移学習はあまり研究されていないため、事前学習済みモデルを利用できるドメインはほとんどありません。例えば、医学の分野では、学習済みのモデルがほとんど存在しないため、転移学習の利用は困難である。また、時系列解析のようなタスクにどのように転移学習を使うかは、まだよく理解されていない。

jargon: Transfer learning: Using a pretrained model for a task different to what it was originally trained for.

専門用語 転移学習： 専門用語：転移学習：事前に学習させたモデルを、元々学習させたものとは異なるタスクに使用すること。

The sixth line of our code tells fastai how to *fit* the model:

このコードの 6 行目は、fastai にモデルの適合を指示するものです：

```
learn.fine_tune(1)
```

As we've discussed, the architecture only describes a *template* for a mathematical function; it doesn't actually do anything until we provide values for the millions of parameters it contains.

これまで述べてきたように、アーキテクチャは数学的関数のテンプレートを記述しているに過ぎず、それが含む何百万ものパラメータに値を与えるまでは、実際には何もしません。

This is the key to deep learning—determining how to fit the parameters of a model to get it to solve your problem. In order to fit a model, we have to provide at least one piece of information: how many times to look at each image (known as number of *epochs*). The number of epochs you select will largely depend on how much time you have available, and how long you find it takes in practice to fit your model. If you select a number that is too small, you can always train for more epochs later.

これが深層学習の鍵で、モデルのパラメータをどのように適合させれば、問題を解決できるかを判断するのです。モデルを適合させるためには、少なくとも 1 つの情報を提供する必要があります：各画像を何回見るか（エポック数）。エポック数は、利用できる時間の長さと、実際にモデルをフィットさせるのにかかる時間によって大きく変わります。もし小さすぎる数を選択した場合は、後でいつでもより多くのエポック数を訓練することができます。

But why is the method called *fine_tune*, and not *fit*? fastai actually *does* have a method called *fit*, which does indeed fit a model (i.e. look at images in the training set multiple times, each time updating the parameters to make the predictions closer and closer to the target labels). But in this case, we've started with a pretrained model, and we don't want to throw away all those capabilities that it already has. As you'll learn in this book, there are some important tricks to adapt a pretrained model for a new dataset—a process called *fine-tuning*.

fastai には *fit* というメソッドがあり、これは確かにモデルのフィッティングを行います（トレーニングセットの画像を何度も見て、その都度パラメータを更新して予測値をターゲット

トラブルに近づけていきます）。しかし、この場合、事前に学習させたモデルから始めているので、そのモデルがすでに持っている能力をすべて捨てたくはありません。本書で学ぶように、事前学習したモデルを新しいデータセットに適応させるには、微調整と呼ばれる重要なトリックがあります。

jargon: Fine-tuning: A transfer learning technique where the parameters of a pretrained model are updated by training for additional epochs using a different task to that used for pretraining.

専門用語です： フайнチューニング： 事前学習したモデルのパラメータを、事前学習に使用したタスクとは異なるタスクを使用して追加のエポック学習を行うことで更新する伝達学習の手法。

When you use the `fine_tune` method, `fastai` will use these tricks for you. There are a few parameters you can set (which we'll discuss later), but in the default form shown here, it does two steps:

`fine_tune` メソッドを使用すると、`fastai` がこれらのトリックを代わりに使ってくれます。設定できるパラメータはいくつかありますが（これについては後述します）、ここに示すデフォルトの形では、2つのステップを実行します：

1. Use one epoch to fit just those parts of the model necessary to get the new random head to work correctly with your dataset.
1回のエポックで、新しいランダムヘッドをデータセットに正しく作用させるために必要なモデルの部分のみを適合させます。
2. Use the number of epochs requested when calling the method to fit the entire model, updating the weights of the later layers (especially the head) faster than the earlier layers (which, as we'll see, generally don't require many changes from the pretrained weights).

メソッドを呼び出す際に要求されたエポック数を使用して、モデル全体をフィットさせ、前の層よりも後の層（特に頭部）の重みを速く更新します（後述するように、一般的に事前学習した重みから多くの変更を必要としません）。

The *head* of a model is the part that is newly added to be specific to the new dataset. An *epoch* is one complete pass through the dataset. After calling `fit`, the results after each epoch are printed, showing the epoch number, the training and validation set losses (the "measure of performance" used for training the model), and any *metrics* you've requested (error rate, in this case).

モデルの頭部は、新しいデータセットに対応するために新たに追加される部分です。エポックとは、データセットを完全に通過する1回のことです。`fit` を呼び出すと、各エポック後の結果が出力され、エポック番号、トレーニングセットと検証セットの損失（モデルのトレーニングに使用された「性能の尺度」）、そしてリクエストしたメトリクス（この場合はエラー率）が示されます。

So, with all this code our model learned to recognize cats and dogs just from labeled examples. But how did it do it?

このようなコードで、このモデルは、ラベル付きの例から猫と犬を認識することを学びました。しかし、どうやってそれを実現したのでしょうか？

What Our Image Recognizer Learned

画像認識エンジンが学習したこと

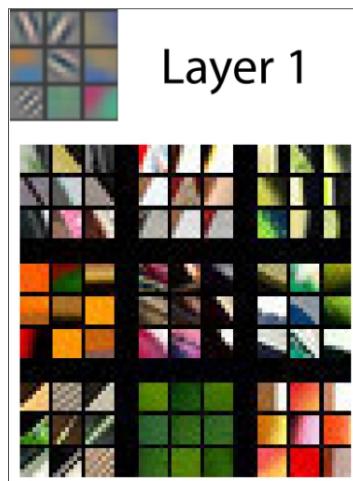
At this stage we have an image recognizer that is working very well, but we have no idea what it is actually doing! Although many people complain that deep learning results in impenetrable "black box" models (that is, something that gives predictions but that no one can understand), this really couldn't be further from the truth. There is a vast body of research showing how to deeply inspect deep learning models, and get rich insights from them. Having said that, all kinds of machine learning models (including deep learning, and traditional statistical models) can be challenging to fully understand, especially when considering how they will behave when coming across data that is very different to the data used to train them. We'll be discussing this issue throughout this book.

この段階では、非常にうまく動作している画像認識装置がありますが、それが実際に何をしているのかはわかりません！多くの人が、深層学習は不可解な「ブラックボックス」モデル（つまり、予測はできるが誰も理解できないもの）になると不満を持っていますが、これは本当に真実から遠く離れたことではありません。ディープラーニングモデルを深く観察し、そこから豊かな洞察を得る方法を示す膨大な研究結果があるのです。とはいえ、あらゆる種類の機械学習モデル（深層学習や従来の統計モデルを含む）を完全に理解するのは難しいものです。特に、学習に使用したデータとはまったく異なるデータに遭遇したときにどのような挙動を示すかを考えると、なおさらです。この問題については、本書を通して議論していくことにします。

In 2013 a PhD student, Matt Zeiler, and his supervisor, Rob Fergus, published the paper "[Visualizing and Understanding Convolutional Networks](#)", which showed how to visualize the neural network weights learned in each layer of a model. They carefully analyzed the model that won the 2012 ImageNet competition, and used this analysis to greatly improve the model, such that they were able to go on to win the 2013 competition!

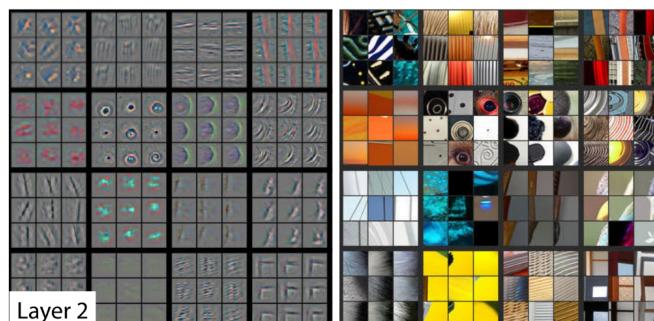
<<img_layer1>> is the picture that they published of the first layer's weights.

2013年、博士課程の学生である Matt Zeiler とその指導教官である Rob Fergus は、モデルの各層で学習されたニューラルネットワークの重みを可視化する方法を示した論文 「Visualizing and Understanding Convolutional Networks」 を発表しました。2012年の ImageNet コンペティションで優勝したモデルを丁寧に分析し、その分析結果をもとにモデルを大幅に改善し、2013年のコンペティションで優勝することができました！<>は、第1層の重みを公開した画像です。



This picture requires some explanation. For each layer, the image part with the light gray background shows the reconstructed weights pictures, and the larger section at the bottom shows the parts of the training images that most strongly matched each set of weights. For layer 1, what we can see is that the model has discovered weights that represent diagonal, horizontal, and vertical edges, as well as various different gradients. (Note that for each layer only a subset of the features are shown; in practice there are thousands across all of the layers.) These are the basic building blocks that the model has learned for computer vision. They have been widely analyzed by neuroscientists and computer vision researchers, and it turns out that these learned building blocks are very similar to the basic visual machinery in the human eye, as well as the handcrafted computer vision features that were developed prior to the days of deep learning. The next layer is represented in <<img_layer2>>.

この図には説明が必要です。各レイヤーについて、背景が薄いグレーの画像部分が再構成された重みの絵で、下部の大きな部分は、トレーニング画像の中で各重みのセットと最も強く一致した部分を示しています。第1層では、斜め、水平、垂直のエッジと、さまざまなグラデーションを表す重みが発見されたことがわかります。(各レイヤーでは特徴の一部しか表示されていませんが、実際にはすべてのレイヤーで何千もの特徴があります)。これらは、コンピュータビジョンのためにモデルが学習した基本的な構成要素である。神経科学者やコンピュータビジョンの研究者によって広く分析され、これらの学習されたビルディングブロックは、人間の目の基本的な視覚機械や、深層学習の時代以前に開発された手作りのコンピュータビジョン機能に非常に似ていることが判明しています。次の層は<<img_layer2>>で表されます。



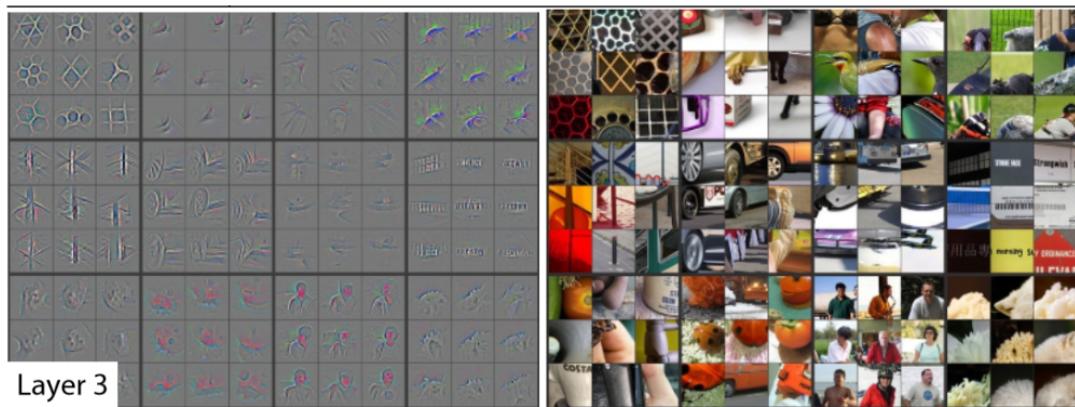
For layer 2, there are nine examples of weight reconstructions for each of the features found by the model. We can see that the model has learned to create feature detectors that look for corners, repeating lines, circles, and other

simple patterns. These are built from the basic building blocks developed in the first layer. For each of these, the right-hand side of the picture shows small patches from actual images which these features most closely match. For instance, the particular pattern in row 2, column 1 matches the gradients and textures associated with sunsets.

<<img_layer3>> shows the image from the paper showing the results of reconstructing the features of layer 3.

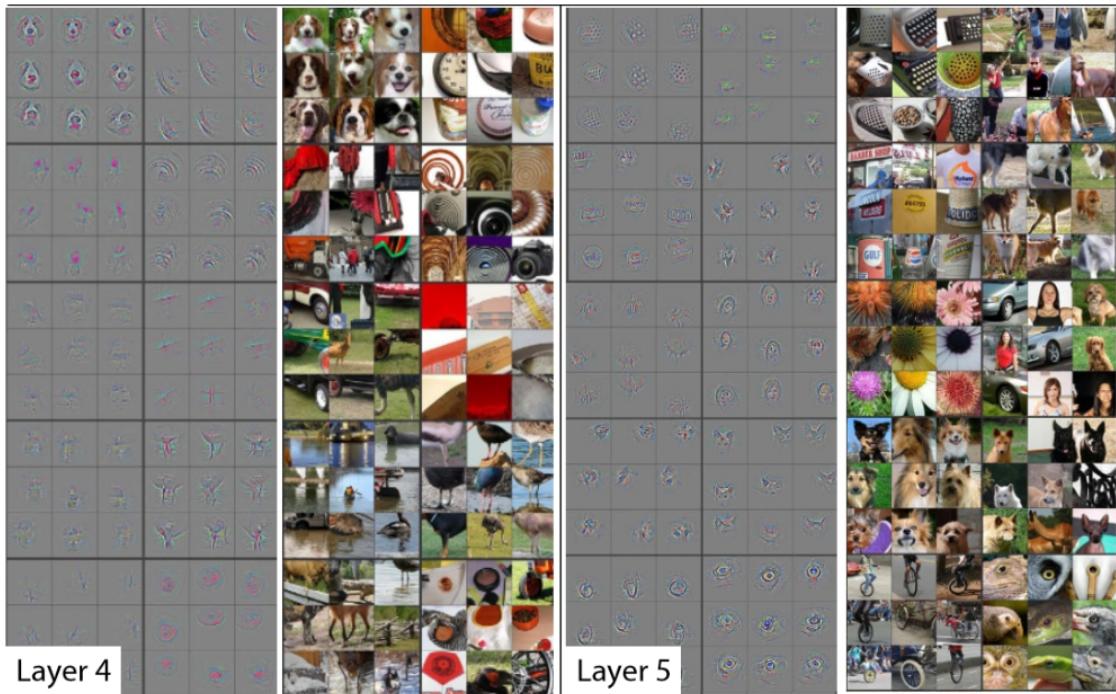
レイヤー2については、モデルによって発見された各特徴に対する重みの再構成の例が9つある。モデルが、角、繰り返しの線、円、その他の単純なパターンを探す特徴検出器を作ることを学習していることがわかる。これらは、第1層で開発された基本的な構成要素から作られています。それぞれの特徴について、右側の図は、これらの特徴が最もよく一致する実際の画像からの小さなパッチを示しています。例えば、2行目1列目のパターンは、夕焼けのグラデーションやテクスチャーにマッチしています。

<<img_layer3>>は、レイヤー3の特徴を再構築した結果を示す論文の画像である。



As you can see by looking at the righthand side of this picture, the features are now able to identify and match with higher-level semantic components, such as car wheels, text, and flower petals. Using these components, layers four and five can identify even higher-level concepts, as shown in <<img_layer4>>.

この絵の右側を見てわかるように、特徴量は、車の車輪、テキスト、花びらなど、より高度な意味成分を識別し、マッチングすることができるようになった。これらの構成要素を用いて、レイヤー4と5は、<<img_layer4>>に示すように、さらに高いレベルの概念を識別することができます。



This article was studying an older model called *AlexNet* that only contained five layers. Networks developed since then can have hundreds of layers—so you can imagine how rich the features developed by these models can be!

この記事は、AlexNet という 5 層しかない古いモデルを研究したものです。それ以降に開発されたネットワークは、何百もの層を持つことができるので、これらのモデルによって開発された特徴がどれほど豊かなものであるかは想像がつくでしょう！

When we fine-tuned our pretrained model earlier, we adapted what those last layers focus on (flowers, humans, animals) to specialize on the cats versus dogs problem. More generally, we could specialize such a pretrained model on many different tasks. Let's have a look at some examples.

先ほど学習済みモデルを微調整した際、最後の層が注目するもの（花、人間、動物）を適応させて、猫と犬の問題に特化させたのです。より一般的には、このような事前学習済みモデルを様々なタスクに特化させることができます。いくつかの例を見てみましょう。

Image Recognizers Can Tackle Non-Image Tasks

画像認識ソフトは画像以外のタスクにも対応できる

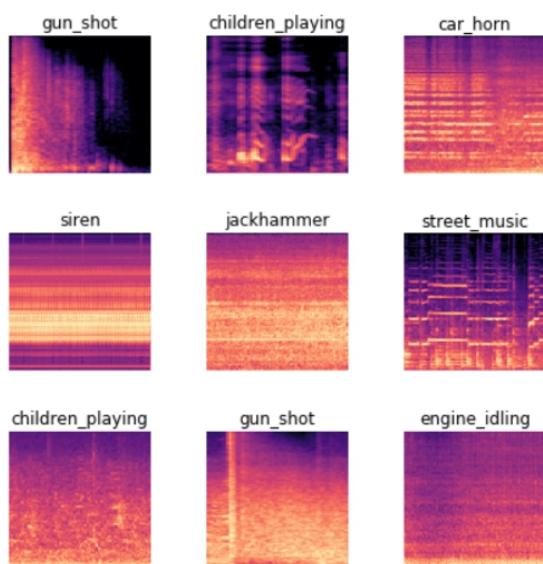
An image recognizer can, as its name suggests, only recognize images. But a lot of things can be represented as images, which means that an image recogniser can learn to complete many tasks.

画像認識装置は、その名の通り、画像だけを認識することができます。しかし、多くのものが画像として表現できるため、画像認識装置は多くのタスクを完了するために学習することができます。

For instance, a sound can be converted to a spectrogram, which is a chart that shows the amount of each frequency at each time in an audio file. Fast.ai student Ethan Sutin used this approach to easily beat the published accuracy of a state-of-the-art [environmental sound detection model](#) using a dataset of 8,732 urban sounds.

fastai's show_batch clearly shows how each different sound has a quite distinctive spectrogram, as you can see in <>.

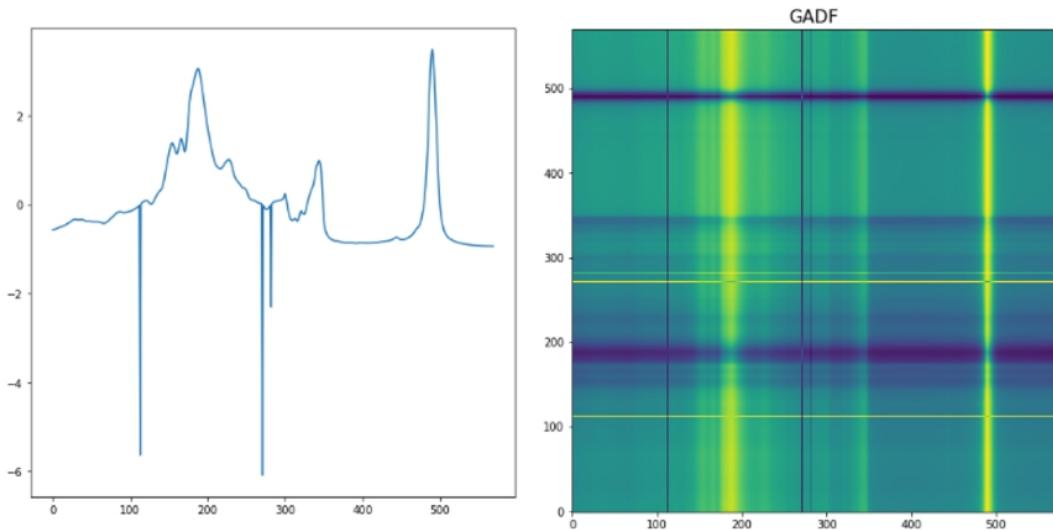
例えば、音はスペクトログラムに変換することができます。スペクトログラムとは、オーディオファイルの各時間における各周波数の量を示すグラフです。Fast.ai の学生である Ethan Sutin は、このアプローチを使って、8,732 の都市音のデータセットを使って、最先端の環境音検出モデルの公表精度を簡単に上回りました。fastai の show_batch では、<> に見られるように、異なる音それぞれが非常に特徴的なスペクトログラムを持っていることがはっきりとわかります。



A time series can easily be converted into an image by simply plotting the time series on a graph. However, it is often a good idea to try to represent your data in a way that makes it as easy as possible to pull out the most important components. In a time series, things like seasonality and anomalies are most likely to be of interest. There are various transformations available for time series data. For instance, fast.ai student Ignacio Oguiza created images from a time series dataset for olive oil classification, using a technique called Gramian Angular Difference Field (GADF); you can see the result in <>. He then fed those images to an image classification model just like the one you see in this chapter. His results, despite having only 30 training set images, were well over 90% accurate, and close to the state of the art.

時系列をグラフにプロットするだけで、簡単に画像化することができます。しかし、できるだけ簡単で重要な要素を引き出せるような方法でデータを表現してみるのがよい場合が多い。時系列では、季節性や異常性といったものが最も注目されやすい。時系列データには、さまざまな変換が

用意されています。例えば、fast.ai の学生である Ignacio Oguiza は、オリーブオイルの分類のための時系列データセットから、Gramian Angular Difference Field(GADF)という手法で画像を作成しました(<<ts_image>>で結果を見るすることができます)。そして、これらの画像を、本章で紹介するような画像分類モデルに与えたのです。その結果、わずか 30 枚のトレーニングセット画像にもかかわらず、90%以上の精度を達成し、最先端の技術に近い結果を得ることができました。



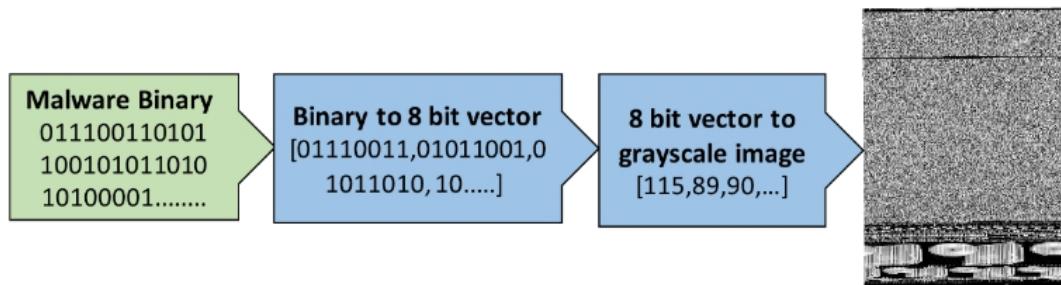
Another interesting fast.ai student project example comes from Gleb Esman. He was working on fraud detection at Splunk, using a dataset of users' mouse movements and mouse clicks. He turned these into pictures by drawing an image where the position, speed, and acceleration of the mouse pointer was displayed using coloured lines, and the clicks were displayed using [small colored circles](#), as shown in <>. He then fed this into an image recognition model just like the one we've used in this chapter, and it worked so well that it led to a patent for this approach to fraud analytics!

もう一つの興味深い fast.ai の学生プロジェクトの例は、Gleb Esman によるものです。彼は Splunk で、ユーザーのマウスの動きとマウスクリックのデータセットを使った不正検知に取り組んでいました。彼は、<>のように、マウスポインタの位置、速度、加速度を色のついた線で表示し、クリックを色のついた小さな円で表示する画像を描くことで、これらを絵に変換しました。これを、この章で使ったのと同じような画像認識モデルにかけると、うまくいったので、この不正分析手法の特許を取得することができました！

Another example comes from the paper "[Malware Classification with Deep Convolutional Neural Networks](#)" by Mahmoud Kalash et al., which explains that "the malware binary file is divided into 8-bit sequences which are then converted to equivalent decimal values. This decimal vector is reshaped and a gray-scale image is generated that represents the malware sample," like in <<malware_proc>>.

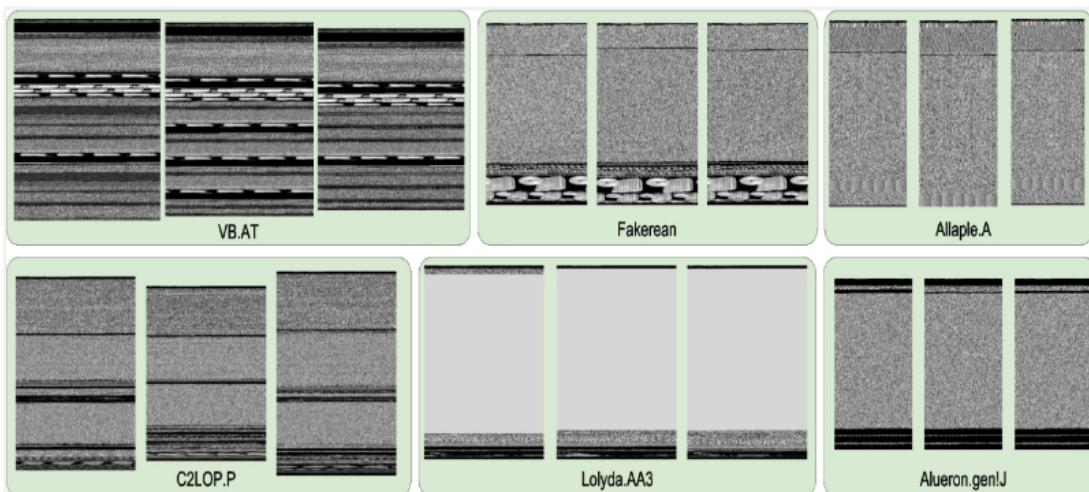
また、Mahmoud Kalash 氏らによる論文「[Malware Classification with Deep Convolutional Neural Networks](#)」によると、「マルウェアバイナリファイルは 8 ビットシーケンスに分割され、それが同等の 10 進数値に変換される」と説明されています。この 10 進数ベクトルは再形成され、マル

「ウェアサンプルを表すグレースケール画像が生成される」と、<<malware_proc>>のように説明されています。



The authors then show "pictures" generated through this process of malware in different categories, as shown in <<malware_eg>>.

そして、このプロセスで生成されたマルウェアの「絵」を、<<malware_eg>>に示すように、さまざまなカテゴリーに分けて表示しています。



As you can see, the different types of malware look very distinctive to the human eye. The model the researchers trained based on this image representation was more accurate at malware classification than any previous approach shown in the academic literature. This suggests a good rule of thumb for converting a dataset into an image representation: if the human eye can recognize categories from the images, then a deep learning model should be able to do so too.

ご覧のように、異なるタイプのマルウェアは、人間の目には非常に特徴的に見えます。研究者がこの画像表現に基づいて学習させたモデルは、学術文献で示されたこれまでのどのアプローチよりも、マルウェア分類の精度が高かったのです。このことは、データセットを画像表現に変換する際の経験則を示唆しています。人間の目が画像からカテゴリを認識できるのであれば、ディープラーニングモデルも同様に認識できるはずです。

In general, you'll find that a small number of general approaches in deep learning can go a long way, if you're a bit creative in how you represent your data! You shouldn't think of approaches like the ones described here as "hacky workarounds," because actually they often (as here) beat previously state-of-the-art results. These really are the right ways to think about these problem domains.

一般的に、ディープラーニングの一般的なアプローチは、データの表現方法を少し工夫することで、長い道のりを歩むことができるようになります！ここで紹介したようなアプローチを「厄介な回避策」と考えるべきではありません。実際、（今回のように）以前の最先端の結果を上回ることがよくあるからです。このような問題領域を考えるには、本当に正しい方法なのです。

Jargon Recap

専門用語のまとめ

We just covered a lot of information so let's recap briefly, <> provides a handy vocabulary.

先ほどは多くの情報を網羅したので簡単におさらいすると、<>は便利なボキャブラリーを提供します。

asciidoc

アスキードック

[[dljargon]]

[[dljargon]]です。

.Deep learning vocabulary

.ディープラーニングボキャブラリー

[options="header"]

[オプション="ヘッダー"]。

|=====

| Term | Meaning

| 用語解説 | 意味

| Label | The data that we're trying to predict, such as "dog" or "cat"

| Label | "犬"や"猫"など、予測しようとするデータ。

| Architecture | The _template_ of the model that we're trying to fit; the actual mathematical function that we're passing the input data and parameters to

| アーキテクチャ | 適合させようとするモデルのひな形、入力データとパラメータを渡す実際の数学関数

| Model | The combination of the architecture with a particular set of parameters

| Model | アーキテクチャに特定のパラメータを組み合わせたもの

| Parameters | The values in the model that change what task it can do, and are updated through model training

| パラメータ | モデルの中で、どのようなタスクができるかを変える値で、モデルのトレーニングを通じて更新される。

| Fit | Update the parameters of the model such that the predictions of the model using the input data match the target labels

| Fit | 入力データを用いたモデルの予測値がターゲットラベルと一致するように、モデルのパラメータを更新する。

| Train | A synonym for _fit_

| トレーニング | _fit_の対義語です。

| Pretrained model | A model that has already been trained, generally using a large dataset, and will be fine-tuned

| プレトレインモデル | 一般に大規模なデータセットを用いて既に学習されたモデルで、今後微調整が行われる。

| Fine-tune | Update a pretrained model for a different task

| Fine-tune | 異なるタスクのために事前学習されたモデルを更新する

| Epoch | One complete pass through the input data

| エポック | 入力データを1回完全に通過させる。

| Loss | A measure of how good the model is, chosen to drive training via SGD

| 損失 | モデルがどの程度優れているかを示す指標で、SGD によるトレーニングを推進するために選択されます。

| Metric | A measurement of how good the model is, using the validation set, chosen for human consumption

| Metric | 人間が消費するために選ばれた検証セットを用いて、モデルがどれだけ優れているかを測定したもの

| Validation set | A set of data held out from training, used only for measuring how good the model is

| バリデーションセット | トレーニングから除外されたデータのセットで、モデルの良し悪しを測るために使用される。

| Training set | The data used for fitting the model; does not include any data from the validation set

| トレーニングセット | モデルのフィッティングに使用するデータで、検証セットのデータは含まれない

| Overfitting | Training a model in such a way that it remembers specific features of the input data, rather than generalizing well to data not seen during training

| オーバーフィット | 入力されたデータの特定の特徴を記憶するようにモデルを学習させる。

| CNN | Convolutional neural network; a type of neural network that works particularly well for computer vision tasks

| CNN | コンボリューション・ニューラル・ネットワーク：コンピュータビジョンのタスクに特に適したニューラルネットワークの一種

| =====

With this vocabulary in hand, we are now in a position to bring together all the key concepts introduced so far. Take a moment to review those definitions and read the following summary. If you can follow the explanation, then you're well equipped to understand the discussions to come.

この語彙を手にすることで、これまで紹介してきた重要な概念をまとめることができるようになりました。ここで、これまでの定義を確認し、次の要約を読んでみてください。この説明で理解できたなら、これから議論を理解するのに十分な能力を持っていると言えます。

Machine learning is a discipline where we define a program not by writing it entirely ourselves, but by learning from data. *Deep learning* is a specialty within machine learning that uses *neural networks* with multiple *layers*. *Image classification* is a representative example (also known as *image recognition*). We start with *labeled data*; that is, a set of images where we have assigned a *label* to each image indicating what it represents. Our goal is to produce a program, called a *model*, which, given a new image, will make an accurate *prediction* regarding what that new image represents.

機械学習は、プログラムを自分で書くのではなく、データから学習して定義する学問である。ディープラーニングは、機械学習の中でも、複数の層を持つニューラルネットワークを使った専門分野です。代表的なものに画像分類があります（画像認識ともいう）。ラベル付けされたデータ、つまり、それぞれの画像に何を表しているかを示すラベルを付けた画像の集合から始めます。我々の目標は、新しい画像が与えられたときに、その画像が何を表しているのかを正確に予測する、モデルと呼ばれるプログラムを作成することである。

Every model starts with a choice of *architecture*, a general template for how that kind of model works internally. The process of *training* (or *fitting*) the model is the process of finding a set of *parameter values* (or *weights*) that specialize that general architecture into a model that works well for our particular kind of data. In order to define how well a model does on a single prediction, we need to define a *loss function*, which determines how we score a prediction as good or bad.

すべてのモデルは、内部でどのように動作するかの一般的なテンプレートであるアーキテクチャを選択することから始まります。モデルのトレーニング（適合）は、一般的なアーキテクチャを、特定の種類のデータに対してうまく機能するモデルに特化させるためのパラメータ値（または重み）のセットを見つけるプロセスである。1つの予測に対してモデルがどの程度うまく機能するかを定義するために、損失関数を定義する必要があります。損失関数は、予測の良し悪しをスコア化する方法です。

To make the training process go faster, we might start with a *pretrained model*—a model that has already been trained on someone else's data. We can then adapt it to our data by training it a bit more on our data, a process called *fine-tuning*.

学習プロセスをより速く進めるために、他の人のデータすでに学習されたモデルから始めることもあります。これは「ファインチューニング（微調整）」と呼ばれるプロセスで、私たちのデータでもう少し訓練することによって、そのモデルを私たちのデータに適合させることができます。

When we train a model, a key concern is to ensure that our model *generalizes*—that is, that it learns general lessons from our data which also apply to new items it will encounter, so that it can make good predictions on those items. The risk is that if we train our model badly, instead of learning general lessons it effectively memorizes what

it has already seen, and then it will make poor predictions about new images. Such a failure is called *overfitting*. In order to avoid this, we always divide our data into two parts, the *training set* and the *validation set*. We train the model by showing it only the training set and then we evaluate how well the model is doing by seeing how well it performs on items from the validation set. In this way, we check if the lessons the model learns from the training set are lessons that generalize to the validation set. In order for a person to assess how well the model is doing on the validation set overall, we define a *metric*. During the training process, when the model has seen every item in the training set, we call that an *epoch*.

モデルを訓練する際に重要なのは、モデルが一般化することです。つまり、モデルがデータから一般的な教訓を学び、それが新たに遭遇する項目にも適用され、その項目に関して正しい予測を行うことができるようになります。しかし、モデルの訓練が不十分だと、一般的な教訓を学ぶ代わりに、すでに見たものを記憶してしまい、新しい画像についてうまく予測できなくなる恐れがあります。このような失敗はオーバーフィッティング（過剰適合）と呼ばれます。これを避けるために、私たちは常にデータをトレーニングセットと検証セットの2つに分けています。トレーニングセットだけを見せてモデルを訓練し、検証セットのアイテムでどの程度うまくいかを見て、モデルの性能を評価します。こうすることで、モデルがトレーニングセットから学んだ教訓が、バリデーションセットに一般化する教訓であるかどうかを確認することができるのです。検証セットでモデルがどの程度うまくいっているかを総合的に評価するために、ある指標を定義します。学習プロセスにおいて、モデルが学習セットのすべての項目を見たとき、それをエポックと呼びます。

All these concepts apply to machine learning in general. That is, they apply to all sorts of schemes for defining a model by training it with data. What makes deep learning distinctive is a particular class of architectures: the architectures based on *neural networks*. In particular, tasks like image classification rely heavily on *convolutional neural networks*, which we will discuss shortly.

これらの概念はすべて、機械学習全般に適用されます。つまり、データを使って学習させることでモデルを定義する、あらゆる方式に適用できる。しかし、ディープラーニングの特徴は、ニューラルネットワークをベースとした特定のアーキテクチャにある。特に、画像分類のようなタスクでは、畳み込みニューラルネットワークに大きく依存しています。

Deep Learning Is Not Just for Image Classification

ディープラーニングは画像分類のためだけではない

Deep learning's effectiveness for classifying images has been widely discussed in recent years, even showing *superhuman* results on complex tasks like recognizing malignant tumors in CT scans. But it can do a lot more than this, as we will show here.

ディープラーニングの画像分類への有効性は近年広く議論されており、CTスキャン中の悪性腫瘍を認識するような複雑なタスクで超人的な結果を示すことさえあります。しかし、ここで紹介するように、これ以外にも多くのことができるのです。

For instance, let's talk about something that is critically important for autonomous vehicles: localizing objects in a picture. If a self-driving car doesn't know where a pedestrian is, then it doesn't know how to avoid one! Creating a model that can recognize the content of every individual pixel in an image is called *segmentation*. Here is how we can train a segmentation model with fastai, using a subset of the [Camvid dataset](#) from the paper "Semantic Object Classes in Video: A High-Definition Ground Truth Database" by Gabruel J. Brostow, Julien Fauqueur, and Roberto Cipolla:

例えば、自律走行車にとって決定的に重要な、写真に写った物体の位置を特定することについて説明しましょう。自動運転車は、歩行者がどこにいるのかわからなければ、どうすれば歩行者を避けられるかわかりません！画像内の個々のピクセルの内容を認識できるモデルを作成することをセグメンテーションと呼びます。ここでは、論文「Semantic Object Classes in Video」の Camvid データセットのサブセットを使って、fastai でセグメンテーションモデルを学習する方法を説明します： Gabruel J. Brostow, Julien Fauqueur, and Roberto Cipolla による論文「Semantic Object Classes in Video: A High-Definition Ground Truth Database」の Camvid データセットの一部を用いて、fastai でセグメンテーションモデルを学習します：

In []:

```
path = untar_data(URLs.CAMVID_TINY)
dls = SegmentationDataLoaders.from_label_func(
    path, bs=8, fnames = get_image_files(path/"images"),
    label_func = lambda o: path/'labels/f{o.stem}_P{o.suffix}',
    codes = np.loadtxt(path/'codes.txt', dtype=str)
)
learn = unet_learner(dls, resnet34)
learn.fine_tune(8)
```

epoch	train_loss	valid_loss	time
0	2.641862	2.140568	00:02

epoch	train_loss	valid_loss	time
0	1.624964	1.464210	00:02
1	1.454148	1.284032	00:02
2	1.342955	1.048562	00:02
3	1.199765	0.852787	00:02
4	1.078090	0.838206	00:02
5	0.975496	0.746806	00:02
6	0.892793	0.725384	00:02
7	0.827645	0.726778	00:02

We are not even going to walk through this code line by line, because it is nearly identical to our previous example! (Although we will be doing a deep dive into segmentation models in <>chapter_arch_details>, along with all of the other models that we are briefly introducing in this chapter, and many, many more.)

このコードは、前の例とほとんど同じなので、一行ずつ見ていくつもりもありません！(この章で簡単に紹介する他のモデルや、もっともっと多くのモデルとともに、<>chapter_arch_details>でセグメンテーションモデルについて深く掘り下げていきますが。)

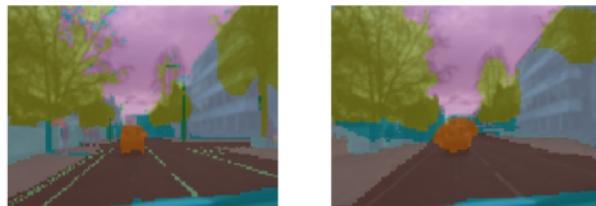
We can visualize how well it achieved its task, by asking the model to color-code each pixel of an image. As you can see, it nearly perfectly classifies every pixel in every object. For instance, notice that all of the cars are overlaid with the same color and all of the trees are overlaid with the same color (in each pair of images, the lefthand image is the ground truth label and the right is the prediction from the model):

画像の各ピクセルを色分けするようモデルに依頼することで、そのタスクがどの程度達成されたかを視覚化することができます。ご覧のように、このモデルはすべてのオブジェクトのすべてのピクセルをほぼ完璧に分類しています。例えば、車はすべて同じ色で、木はすべて同じ色で重なっていることに注目してください(各画像のペアにおいて、左側の画像は真実のラベル、右側はモデルによる予測値です)：

In []:

```
learn.show_results(max_n=6, figsize=(7,8))
```

Target/Prediction



One other area where deep learning has dramatically improved in the last couple of years is natural language processing (NLP). Computers can now generate text, translate automatically from one language to another, analyze comments, label words in sentences, and much more. Here is all of the code necessary to train a model that can classify the sentiment of a movie review better than anything that existed in the world just five years ago:

ディープラーニングがここ数年で劇的に進歩したもう一つの分野は、自然言語処理(NLP)です。コンピュータは現在、テキストの生成、ある言語から別の言語への自動翻訳、コメントの分析、文中の単語のラベル付けなど、さまざまなことができるようになっています。ここでは、わずか5年前に世の中に存在したものより優れた、映画レビューの感情を分類できるモデルを学習するのに必要なコードをすべて紹介します：

In []:

```
from fastai.text.all import *
dls = TextDataLoaders.from_folder(unzip_data(URLs.IMDB), valid='test')
learn = text_classifier_learner(dls, AWD_LSTM, drop_mult=0.5, metrics=accuracy)
learn.fine_tune(4, 1e-2)
```

epoch	train_loss	valid_loss	accuracy	time
0	0.878776	0.748753	0.500400	01:27

epoch	train_loss	valid_loss	accuracy	time
0	0.679118	0.674778	0.584040	02:45
1	0.653671	0.670396	0.618040	02:55
2	0.598665	0.551815	0.718920	05:28
3	0.556812	0.507450	0.752480	03:11

Clean

クリーン

If you hit a "CUDA out of memory error" after running this cell, click on the menu Kernel, then restart. Instead of executing the cell above, copy and paste the following code in it:

このセルを実行した後に「CUDA out of memory error」にぶつかった場合は、メニュー「Kernel」をクリックし、再起動してください。上記のセルを実行する代わりに、次のコードをコピーして貼り付けてください：

```
from fastai.text.all import *
dls = TextDataLoaders.from_folder(unzip_data(URLs.IMDB), valid='test', bs=32)
learn = text_classifier_learner(dls, AWD_LSTM, drop_mult=0.5, metrics=accuracy)
learn.fine_tune(4, 1e-2)
```

This reduces the batch size to 32 (we will explain this later). If you keep hitting the same error, change 32 to 16.

これにより、バッチサイズが32に減少します（これについては後ほど説明します）。同じエラーが続くようであれば、32を16に変更します。

This model is using the "[IMDb Large Movie Review dataset](#)" from the paper "Learning Word Vectors for Sentiment Analysis" by Andrew Maas et al. It works well with movie reviews of many thousands of words, but let's test it out on a very short one to see how it does its thing:

このモデルは、Andrew Maas らの論文「Learning Word Vectors for Sentiment Analysis」の「IMDb Large Movie Review dataset」を使っています。何千語もある映画レビューでもうまくいきますが、非常に短い映画でテストして、その動作を確認してみましょう：

```
In [ ]:  
learn.predict("I really liked that movie!")  
Out[ ]:  
('neg', tensor(0), tensor([0.8786, 0.1214]))
```

Here we can see the model has considered the review to be positive. The second part of the result is the index of "pos" in our data vocabulary and the last part is the probabilities attributed to each class (99.6% for "pos" and 0.4% for "neg").

ここでは、モデルがレビューをポジティブと判断していることがわかります。結果の 2 番目の部分は、データボキャブラリーにおける「pos」のインデックスで、最後の部分は、各クラスに帰属する確率です（「pos」は 99.6%、「neg」は 0.4%）。

Now it's your turn! Write your own mini movie review, or copy one from the internet, and you can see what this model thinks about it.

さて、次はあなたの番です！あなた自身のミニ映画レビューを書くか、インターネットからコピーして、このモデルがそれについてどう思うかを見ることができます。

Sidebar: The Order Matters

サイドバー：順番が重要

In a Jupyter notebook, the order in which you execute each cell is very important. It's not like Excel, where everything gets updated as soon as you type something anywhere—it has an inner state that gets updated each time you execute a cell. For instance, when you run the first cell of the notebook (with the "CLICK ME" comment), you create an object called learn that contains a model and data for an image classification problem. If we were to run the cell just shown in the text (the one that predicts if a review is good or not) straight after, we would get an error as this learn object does not contain a text classification model. This cell needs to be run after the one containing:

Jupyter ノートブックでは、各セルを実行する順番が非常に重要です。Excel のように、どこかに何かを入力するとすぐにすべてが更新されるのではなく、セルを実行するたびに更新される内部状態があります。例えば、ノートブックの最初のセル（「CLICK ME」のコメント付き）を実行すると、画像分類問題のモデルとデータを含む learn というオブジェクトが作成されます。もし、本文にあるセル（レビューの良し悪しを予測するセル）を直後に実行すると、この learn オブジェクトにはテキスト分類モデルが含まれていないため、エラーになります。このセルは、テキスト分類モデルを含むセルの後に実行する必要がある：

```
from fastai.text.all import *
dls = TextDataLoaders.from_folder(unzip_data(URLs.IMDB),
valid='test')learn = text_classifier_learner(dls, AWD_LSTM,
drop_mult=0.5,
metrics=accuracy)learn.fine_tune(4,
1e-2)
```

The outputs themselves can be deceiving, because they include the results of the last time the cell was executed; if you change the code inside a cell without executing it, the old (misleading) results will remain.

出力自体には、前回セルを実行したときの結果が含まれているため、騙される可能性があります。実行せずにセル内のコードを変更すると、古い（誤解を招く）結果が残ります。

Except when we mention it explicitly, the notebooks provided on the [book website](#) are meant to be run in order, from top to bottom. In general, when experimenting, you will find yourself executing cells in any order to go fast (which is a super neat feature of Jupyter Notebook), but once you have explored and arrived at the final version of your code, make sure you can run the cells of your notebooks in order (your future self won't necessarily remember the convoluted path you took otherwise!).

明示的に言及する場合を除き、書籍のウェブサイトで提供されているノートブックは、上から下へ順番に実行することを意図しています。一般的に、実験しているときは、速く進めるためにセルを任意の順序で実行することになるでしょう（これは Jupyter Notebook の非常に優れた機能です）が、一度探求してコードの最終バージョンに到達したら、ノートブックのセルを順番に実行できることを確認してください（そうしないと、未来の自分は必ずしもあなたが取った複雑な経路を覚えていないでしょう！）。

In command mode, pressing 0 twice will restart the *kernel* (which is the engine powering your notebook). This will wipe your state clean and make it as if you had just started in the notebook. Choose Run All Above from the Cell menu to run all cells above the point where you are. We have found this to be very useful when developing the fastai library.

コマンドモードで 0 を 2 回押すと、カーネル（ノートブックの動力源となるエンジン）が再起動します。これにより、あなたの状態は一掃され、あたかもノートブックを始めたばかりのようになります。Cell メニューから Run All Above を選択すると、現在地より上のすべてのセルを実行することができます。これは、fastai ライブラリを開発する際に非常に便利な機能であることがわかりました。

End sidebar

サイドバーを終了する

If you ever have any questions about a fastai method, you should use the function `doc`, passing it the method name:

fastai のメソッドについて質問がある場合は、メソッド名を渡して、関数 doc を使用する必要があります:

```
doc(learn.predict)
```

This will make a small window pop up with content like this:

そうすると、このような内容の小窓がポップアップ表示されます:

The screenshot shows a Jupyter Notebook cell with the following content:

```
Learner.predict
```

`Learner.predict(item, rm_type_tfms=None, with_input=False)`

Return the prediction on `item`, fully decoded, loss function decoded and probabilities
[Show in docs](#)

A brief one-line explanation is provided by doc. The "Show in docs" link takes you to the full documentation, where you'll find all the details and lots of examples. Also, most of fastai's methods are just a handful of lines, so you can click the "source" link to see exactly what's going on behind the scenes.

簡単な一行説明が doc.Show in docs で提供されます。Show in docs リンクをクリックすると、完全なドキュメントが表示され、すべての詳細と多くの例を見るることができます。また、fastai のメソッドのほとんどはほんの数行なので、「ソース」リンクをクリックすれば、舞台裏で何が起こっているのかを正確に見ることができます。

Let's move on to something much less sexy, but perhaps significantly more widely commercially useful: building models from plain *tabular* data.

次に、あまりセクシーではないが、おそらくもっと広く商業的に有用なもの、すなわち、普通の表形式のデータからモデルを構築することに移ろう。

jargon: Tabular: Data that is in the form of a table, such as from a spreadsheet, database, or CSV file. A tabular model is a model that tries to predict one column of a table based on information in other columns of the table.

専門用語 表形式: 表計算ソフト、データベース、CSV ファイルなど、表の形をしたデータ。表形式モデルとは、表の他の列の情報に基づいて、表のある列を予測しようとするモデルである。

It turns out that looks very similar too. Here is the code necessary to train a model that will predict whether a person is a high-income earner, based on their socioeconomic background:

これも非常によく似ていることがわかります。以下は、社会経済的背景に基づいて、ある人が高所得者であるかどうかを予測するモデルを学習するのに必要なコードです:

In []:

```

from fastai.tabular.all import * path = untar_data(URLs.ADULT_SAMPLE) dls =
TabularDataLoaders.from_csv(path/'adult.csv', path=path, y_names="salary", cat_names = ['workclass', 'education',
'marital-status', 'occupation', 'relationship', 'race'], cont_names = ['age', 'fnlwgt', 'education-num'], procs =
[Categorify, FillMissing, Normalize]) learn = tabular_learner(dls, metrics=accuracy)

```

As you see, we had to tell fastai which columns are *categorical* (that is, contain values that are one of a discrete set of choices, such as occupation) and which are *continuous* (that is, contain a number that represents a quantity, such as age).

ご覧のように、fastai にどの列がカテゴリカル（つまり、職業などの離散的な選択肢の一つである値を含む）で、どの列が連続（つまり、年齢などの量を表す数字を含む）であるかを伝える必要がありました。

There is no pretrained model available for this task (in general, pretrained models are not widely available for any tabular modeling tasks, although some organizations have created them for internal use), so we don't use fine_tune in this case. Instead we use fit_one_cycle, the most commonly used method for training fastai models *from scratch* (i.e. without transfer learning):

このタスクで利用できる事前学習済みモデルはありません（一般に、事前学習済みモデルは、どの表モデリングタスクでも広く利用できるわけではありません。その代わりに、fastai モデルをゼロから（つまり伝達学習なしで）学習するための最も一般的な方法である fit_one_cycle を使用します：

In []:

```
learn.fit_one_cycle(3)
```

epoch	train_loss	valid_loss	accuracy	time
0	0.372397	0.357177	0.832463	00:08
1	0.351544	0.341505	0.841523	00:08
2	0.338763	0.339184	0.845670	00:08

This model is using the [Adult dataset](#), from the paper "Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid" by Rob Kohavi, which contains some demographic data about individuals (like their education, marital status, race, sex, and whether or not they have an annual income greater than \$50k). The model is over 80% accurate, and took around 30 seconds to train.

このモデルは、Rob Kohaviによる論文「Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid」の Adult データセットを使用しており、個人に関するいくつかの人口統計データ（教育、婚姻状況、人種、性別、年間所得が5万ドル以上かどうかなど）を含んでいます。このモデルの精度は 80%以上で、訓練にかかった時間は約 30 秒です。

Let's look at one more. Recommendation systems are very important, particularly in e-commerce. Companies like Amazon and Netflix try hard to recommend products or movies that users might like. Here's how to train a model that will predict movies people might like, based on their previous viewing habits, using the [MovieLens dataset](#):

もう1つ見てみましょう。レコメンデーションシステムは、特に電子商取引において非常に重要なです。AmazonやNetflixのような企業は、ユーザーが好きそうな商品や映画を一生懸命に推薦しています。ここでは、MovieLensデータセットを使って、過去の視聴習慣から、ユーザーが好きそうな映画を予測するモデルを訓練する方法を紹介します：

In []:

```
from fastai.collab import *
path = untar_data(URLs.ML_SAMPLE)
dls = CollabDataLoaders.from_csv(path/'ratings.csv')
learn = collab_learner(dls, y_range=(0.5,5.5))
learn.fine_tune(10)
```

epoch	train_loss	valid_loss	time
0	1.510897	1.410028	00:00
epoch	train_loss	valid_loss	time
0	1.375435	1.350930	00:00
1	1.270062	1.173962	00:00
2	1.023159	0.879298	00:00
3	0.797398	0.739787	00:00
4	0.685500	0.700903	00:00
5	0.646508	0.686387	00:00
6	0.623985	0.681087	00:00
7	0.606319	0.676885	00:00
8	0.606975	0.675833	00:00
9	0.602670	0.675682	00:00

This model is predicting movie ratings on a scale of 0.5 to 5.0 to within around 0.6 average error. Since we're predicting a continuous number, rather than a category, we have to tell fastai what range our target has, using the `y_range` parameter.

このモデルは、0.5から5.0のスケールで映画の評価を平均0.6程度の誤差で予測するものです。カテゴリではなく、連続した数値を予測しているので、`y_range` パラメータを使って、ターゲットがどのような範囲を持つかを fastai に伝える必要があります。

Although we're not actually using a pretrained model (for the same reason that we didn't for the tabular model), this example shows that fastai lets us use `fine_tune` anyway in this case (you'll learn how and why this works in <>). Sometimes it's best to experiment with `fine_tune` versus `fit_one_cycle` to see which works best for your dataset.

表モデルと同じ理由で) 実際には事前学習されたモデルを使用していませんが、この例では fastai が fine_tune を使用できることを示しています(この仕組みと理由は <<chapter_pet_breeds>> で学ぶことができます)。fine_tune と fit_one_cycle のどちらがデータセットに最適か、試してみるのが一番いい場合もあります。

We can use the same show_results call we saw earlier to view a few examples of user and movie IDs, actual ratings, and predictions:

先ほどと同じ show_results コールを使って、ユーザーと映画の ID、実際の評価、そして予測の例をいくつか表示することができます：

In []:

```
learn.show_results()
```

	userId	movieId	rating	rating_pred
0	66.0	79.0	4.0	3.978900
1	97.0	15.0	4.0	3.851795
2	55.0	79.0	3.5	3.945623
3	98.0	91.0	4.0	4.458704
4	53.0	7.0	5.0	4.670005
5	26.0	69.0	5.0	4.319870
6	81.0	16.0	4.5	4.426761
7	80.0	7.0	4.0	4.046183
8	51.0	94.0	5.0	3.499996

Sidebar: Datasets: Food for Models

サイドバー：データセット モデルのための食料

You've already seen quite a few models in this section, each one trained using a different dataset to do a different task. In machine learning and deep learning, we can't do anything without data. So, the people that create datasets for us to train our models on are the (often underappreciated) heroes. Some of the most useful and important datasets are those that become important *academic baselines*; that is, datasets that are widely studied by researchers and used to compare algorithmic changes. Some of these become household names (at least, among households that train models!), such as MNIST, CIFAR-10, and ImageNet.

このセクションでは、すでにかなりの数のモデルを見てきましたが、それぞれ異なるタスクのために異なるデータセットを使って訓練されたものです。機械学習や深層学習では、データがなければ何もできません。ですから、モデルを訓練するためのデータセットを作ってくれる人たちは、（しばしば過小評価される）ヒーローなのです。最も有用で重要な

データセットの中には、重要な学術的ベースラインとなるものがあります。つまり、研究者によって広く研究され、アルゴリズムの変更を比較するために使用されるデータセットということです。MNIST、CIFAR-10、ImageNetなど、（少なくとも、モデルを訓練する家庭の間では）有名なデータセットもあります。

The datasets used in this book have been selected because they provide great examples of the kinds of data that you are likely to encounter, and the academic literature has many examples of model results using these datasets to which you can compare your work.

本書で使用するデータセットは、あなたが遭遇する可能性が高い種類のデータの素晴らしい例を提供し、学術文献にはこれらのデータセットを使用したモデル結果の例が多数あり、あなたの作業を比較することができるため、選択されました。

Most datasets used in this book took the creators a lot of work to build. For instance, later in the book we'll be showing you how to create a model that can translate between French and English. The key input to this is a French/English parallel text corpus prepared back in 2009 by Professor Chris Callison-Burch of the University of Pennsylvania. This dataset contains over 20 million sentence pairs in French and English. He built the dataset in a really clever way: by crawling millions of Canadian web pages (which are often multilingual) and then using a set of simple heuristics to transform URLs of French content onto URLs pointing to the same content in English.

本書で使用されているほとんどのデータセットは、作成者が多くの労力をかけて作成したもので、例えば、本書の後半では、フランス語と英語の翻訳を行うモデルの作り方を紹介します。そのための重要なインプットが、ペンシルベニア大学の Chris Callison-Burch 教授が 2009 年に作成した仏英並行テキストコーパスです。このデータセットには、フランス語と英語の 2,000 万以上の文のペアが含まれています。このデータセットは、何百万ものカナダのウェブページ（多言語であることが多い）をクロールし、簡単なヒューリスティックを使って、フランス語のコンテンツの URL を、英語の同じコンテンツを指す URL に変換するという、非常に賢い方法で作られました。

As you look at datasets throughout this book, think about where they might have come from, and how they might have been curated. Then think about what kinds of interesting datasets you could create for your own projects.
(We'll even take you step by step through the process of creating your own image dataset soon.)

本書を通してデータセットを見るとき、それらがどこから来たのか、どのようにキュレーションされたのかを考えてみてください。そして、自分のプロジェクトのために、どのような興味深いデータセットを作ることができるか考えてみてください。(近いうちに、あなたの自身の画像データセットを作成するプロセスをステップバイステップで紹介します)

fast.ai has spent a lot of time creating cut-down versions of popular datasets that are specially designed to support rapid prototyping and experimentation, and to be easier to learn with. In this book we will often start by using one of the cut-down versions and later scale up to the full-size version (just as we're doing in this chapter!). In fact, this is how the world's top practitioners do their modeling in practice; they do most of their experimentation and prototyping with subsets of their data, and only use the full dataset when they have a good understanding of what they have to do.

fast.aiは、人気のあるデータセットのカットダウンバージョンを作成することに多くの時間を費やしてきました。これは、迅速なプロトタイピングや実験をサポートし、学習しやすいように特別に設計されています。本書では、まずカットダウンバージョンの1つを使い、後でフルサイズバージョンにスケールアップすることがよくあります（本章でやっているのと同じです！）。実際、世界のトップレベルの実務家たちは、このようにしてモデリングを実践しています。彼らは、実験やプロトタイピングのほとんどをデータのサブセットで行い、何をすべきかを十分に理解したときに初めてフルデータセットを使用するのです。

End sidebar

サイドバーを閉じる

Each of the models we trained showed a training and validation loss. A good validation set is one of the most important pieces of the training process. Let's see why and learn how to create one.

トレーニングした各モデルには、トレーニングとバリデーションのロスが表示されました。良い検証セットは、トレーニングプロセスで最も重要なものの1つです。その理由と作成方法について説明します。

Validation Sets and Test Sets

バリデーションセットとテストセット

As we've discussed, the goal of a model is to make predictions about data. But the model training process is fundamentally dumb. If we trained a model with all our data, and then evaluated the model using that same data, we would not be able to tell how well our model can perform on data it hasn't seen. Without this very valuable piece of information to guide us in training our model, there is a very good chance it would become good at making predictions about that data but would perform poorly on new data.

これまで述べてきたように、モデルの目的は、データに関する予測を行うことです。しかし、モデルのトレーニングプロセスは、根本的に間抜けです。もし、すべてのデータを使ってモデルをトレーニングし、同じデータを使ってモデルを評価した場合、そのモデルが見たことのないデータに対してどの程度のパフォーマンスを発揮できるかを知ることはできません。このような貴重な情報がなければ、モデルを学習させる際に、そのデータについての予測は得意でも、新しいデータについてはうまくいかない可能性が非常に高くなります。

To avoid this, our first step was to split our dataset into two sets: the *training set* (which our model sees in training) and the *validation set*, also known as the *development set* (which is used only for evaluation). This lets us test that the model learns lessons from the training data that generalize to new data, the validation data.

このような事態を避けるため、私たちはまず、データセットを2つのセットに分割しました。トレーニングセット（モデルがトレーニングで使用するセット）とバリデーションセット（開発セットとも呼ばれる、評価にのみ使用するセット）です。これにより、モデルがトレーニングデータから得た教訓を、新たなデータである検証データに対して一般化できるかどうかを検証することができます。

One way to understand this situation is that, in a sense, we don't want our model to get good results by "cheating." If it makes an accurate prediction for a data item, that should be because it has learned characteristics of that kind of item, and not because the model has been shaped by *actually having seen that particular item*.

この状況を理解する 1 つの方法は、ある意味、モデルが "ズル" をして良い結果を得ることを望んでいない、ということです。あるデータについて正確な予測を行う場合、それはそのデータについての特徴を学習したためであり、実際にそのデータを見たことによってモデルが形成されたためではないはずです。

Splitting off our validation data means our model never sees it in training and so is completely untainted by it, and is not cheating in any way. Right?

検証用データを切り離すということは、モデルがトレーニングでそのデータを見ることがないため、全く汚染されないということであり、何ら不正行為ではありません。そうでしょう？

In fact, not necessarily. The situation is more subtle. This is because in realistic scenarios we rarely build a model just by training its weight parameters once. Instead, we are likely to explore many versions of a model through various modeling choices regarding network architecture, learning rates, data augmentation strategies, and other factors we will discuss in upcoming chapters. Many of these choices can be described as choices of *hyperparameters*. The word reflects that they are parameters about parameters, since they are the higher-level choices that govern the meaning of the weight parameters.

実は、そうとは限りません。状況はもっと微妙なのです。というのも、現実的なシナリオでは、重みパラメータを 1 回学習させるだけでモデルを構築することはほとんどないからです。その代わり、ネットワーク・アーキテクチャ、学習速度、データ増強戦略など、次章で説明するさまざまなモデリングの選択を通じて、何種類ものモデルのバージョンを検討することになります。これらの選択の多くは、ハイパーパラメータの選択と表現できる。この言葉は、重みパラメータの意味を支配する上位の選択であるため、パラメータに関するパラメータであることを反映しています。

The problem is that even though the ordinary training process is only looking at predictions on the training data when it learns values for the weight parameters, the same is not true of us. We, as modelers, are evaluating the model by looking at predictions on the validation data when we decide to explore new hyperparameter values! So subsequent versions of the model are, indirectly, shaped by us having seen the validation data. Just as the automatic training process is in danger of overfitting the training data, we are in danger of overfitting the validation data through human trial and error and exploration.

問題は、通常の学習プロセスでは、重みパラメータの値を学習する際に、学習データに対する予測値しか見ていないにもかかわらず、私たちには同じことが当てはまらないことがあります。私たちモデルーは、新しいハイパーパラメータの値を探索することを決定したとき、検証データの予測値を見てモデルを評価するのです！つまり、その後のモデルのバージョンは、私たちが検証データを見たことによって、間接的に形作られているのです。自動トレーニングプロセスがトレーニングデータをオーバーフィットさせる危険性があるよう

に、私たちも人間の試行錯誤や探索によって検証データをオーバーフィットさせる危険性があるのです。

The solution to this conundrum is to introduce another level of even more highly reserved data, the *test set*. Just as we hold back the validation data from the training process, we must hold back the test set data even from ourselves. It cannot be used to improve the model; it can only be used to evaluate the model at the very end of our efforts. In effect, we define a hierarchy of cuts of our data, based on how fully we want to hide it from training and modeling processes: training data is fully exposed, the validation data is less exposed, and test data is totally hidden. This hierarchy parallels the different kinds of modeling and evaluation processes themselves—the automatic training process with back propagation, the more manual process of trying different hyper-parameters between training sessions, and the assessment of our final result.

この難問を解決するためには、さらに高度な予約データであるテストセットをもう一段階導入することです。検証データをトレーニングプロセスから遠ざけるのと同様に、テストセットのデータも自分自身から遠ざけなければなりません。モデルを改良するために使うことはできず、努力の最後にモデルを評価するためにのみ使うことができる。つまり、トレーニングやモデリングプロセスからどの程度データを隠したいかによって、データをカットする階層を定義しているのです。この階層は、バックプロパゲーションによる自動トレーニングプロセス、トレーニングセッションの間に異なるハイパーパラメータを試すより手動的なプロセス、そして最終結果の評価という、さまざまな種類のモデリングと評価プロセスそのものに類似している。

The test and validation sets should have enough data to ensure that you get a good estimate of your accuracy. If you're creating a cat detector, for instance, you generally want at least 30 cats in your validation set. That means that if you have a dataset with thousands of items, using the default 20% validation set size may be more than you need. On the other hand, if you have lots of data, using some of it for validation probably doesn't have any downsides.

テストセットと検証セットには、精度を正しく見積もるのに十分なデータが必要です。例えば、猫検出器を作成する場合、一般的に、検証セットには少なくとも 30 匹の猫が必要です。つまり、何千ものアイテムがあるデータセットでは、デフォルトの 20% の検証セットサイズを使用すると、必要以上になる可能性があります。一方、たくさんのデータがある場合、その一部を検証用に使っても、おそらくデメリットはないでしょう。

Having two levels of "reserved data"—a validation set and a test set, with one level representing data that you are virtually hiding from yourself—may seem a bit extreme. But the reason it is often necessary is because models tend to gravitate toward the simplest way to do good predictions (memorization), and we as fallible humans tend to gravitate toward fooling ourselves about how well our models are performing. The discipline of the test set helps us keep ourselves intellectually honest. That doesn't mean we *always* need a separate test set—if you have very little data, you may need to just have a validation set—but generally it's best to use one if at all possible.

検証セットとテストセットという 2 つのレベルの「予約データ」を持ち、1 つのレベルは事実上自分自身から隠しているデータを表すというのは、少し極端に思えるかもしれません。しかし、これが必要な理由は、モデルは優れた予測を行うための最も単純な方法（記憶）

に引き寄せられる傾向があり、私たち誤りを犯しやすい人間は、モデルがどの程度うまく機能しているかを自分自身で騙すことに引き寄せられる傾向があるためです。テストセットという規律は、私たちが知的好奇心を保つのに役立ちます。データが少ない場合は、検証セットを用意すればよいのですが、一般的には、可能な限りテストセットを使用するのがベストです。

This same discipline can be critical if you intend to hire a third party to perform modeling work on your behalf. A third party might not understand your requirements accurately, or their incentives might even encourage them to misunderstand them. A good test set can greatly mitigate these risks and let you evaluate whether their work solves your actual problem.

このような規律は、モデリング作業を代行するサードパーティを雇う場合にも重要です。サードパーティは、お客様の要件を正確に理解していないかもしれませんし、インセンティブによって要件を誤解してしまうかもしれません。優れたテストセットは、こうしたリスクを大幅に軽減し、第三者の仕事が実際の問題を解決しているかどうかを評価することができます。

To put it bluntly, if you're a senior decision maker in your organization (or you're advising senior decision makers), the most important takeaway is this: if you ensure that you really understand what test and validation sets are and why they're important, then you'll avoid the single biggest source of failures we've seen when organizations decide to use AI. For instance, if you're considering bringing in an external vendor or service, make sure that you hold out some test data that the vendor *never gets to see*. Then *you* check their model on your test data, using a metric that *you* choose based on what actually matters to you in practice, and *you* decide what level of performance is adequate. (It's also a good idea for you to try out some simple baseline yourself, so you know what a really simple model can achieve. Often it'll turn out that your simple model performs just as well as one produced by an external "expert"!)

率直に言って、もしあなたが組織の上級意思決定者であるなら（あるいは上級意思決定者に助言しているのなら）、最も重要なことは、テストセットと検証セットとは何か、なぜそれが重要なのかを本当に理解していれば、組織がAIを使うことにしたときに見られる最大の失敗の原因を避けることができる、ということです。例えば、外部のベンダーやサービスを導入することを検討している場合、ベンダーが決して見ることのできないテストデータを用意しておくようにします。そして、そのテストデータで、実際に実務で重要なことに基づいて選んだ指標で彼らのモデルをチェックし、どの程度の性能が適切かを判断するのです。（自分でも簡単なベースラインを試してみて、本当に簡単なモデルで何ができるかを知っておくのもよいでしょう。自分の作ったシンプルなモデルが、外部の"専門家"が作ったものと同じような性能を発揮することもよくあることです！）

Use Judgment in Defining Test Sets

テストセットの定義に判断力を働かせる

To do a good job of defining a validation set (and possibly a test set), you will sometimes want to do more than just randomly grab a fraction of your original dataset. Remember: a key property of the validation and test sets is that they must be representative of the new data you will see in the future. This may sound like an impossible order! By definition, you haven't seen this data yet. But you usually still do know some things.

検証セット（場合によってはテストセット）をうまく定義するためには、元のデータセットの一部をランダムに取ってくるだけでは不十分なことがあります。覚えておいてほしいのは、検証セットとテストセットの重要な特性は、将来見ることになる新しいデータを代表するものでなければならないということです。これは不可能な注文のように聞こえるかもしれません！なぜなら、あなたはまだそのデータを見たことがないからです。しかし、通常、あなたはまだいくつかのことを知っているのです。

It's instructive to look at a few example cases. Many of these examples come from predictive modeling competitions on the [Kaggle](#) platform, which is a good representation of problems and methods you might see in practice.

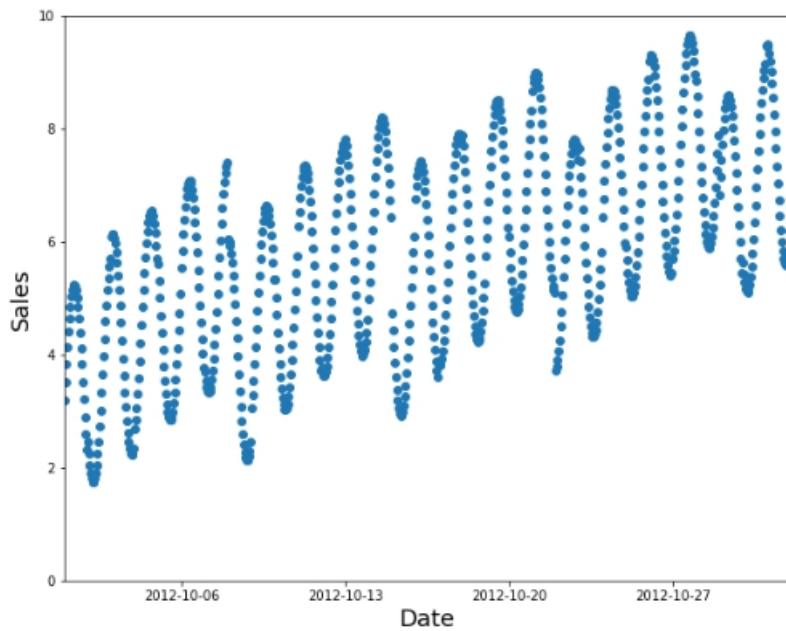
いくつかの事例を見ると、参考になることがあります。これらの事例の多くは、Kaggle プラットフォームの予測モデリング・コンペティションから得られたもので、実際に目にする可能性のある問題や手法をよく表現しています。

One case might be if you are looking at time series data. For a time series, choosing a random subset of the data will be both too easy (you can look at the data both before and after the dates you are trying to predict) and not representative of most business use cases (where you are using historical data to build a model for use in the future). If your data includes the date and you are building a model to use in the future, you will want to choose a continuous section with the latest dates as your validation set (for instance, the last two weeks or last month of available data).

例えば、時系列データを見ている場合です。時系列データの場合、データのランダムなサブセットを選ぶのは簡単すぎるし（予測しようとする日付の前後両方のデータを見ることができる）、ほとんどのビジネスユースケース（将来使用するモデルを構築するために過去のデータを使用する場合）を代表するものではないでしょう。データに日付が含まれていて、将来使用するモデルを構築する場合、検証セットとして最新の日付を持つ連続したセクションを選択することをお勧めします（例えば、利用可能なデータの最後の 2 週間または最後の 1 ヶ月）。

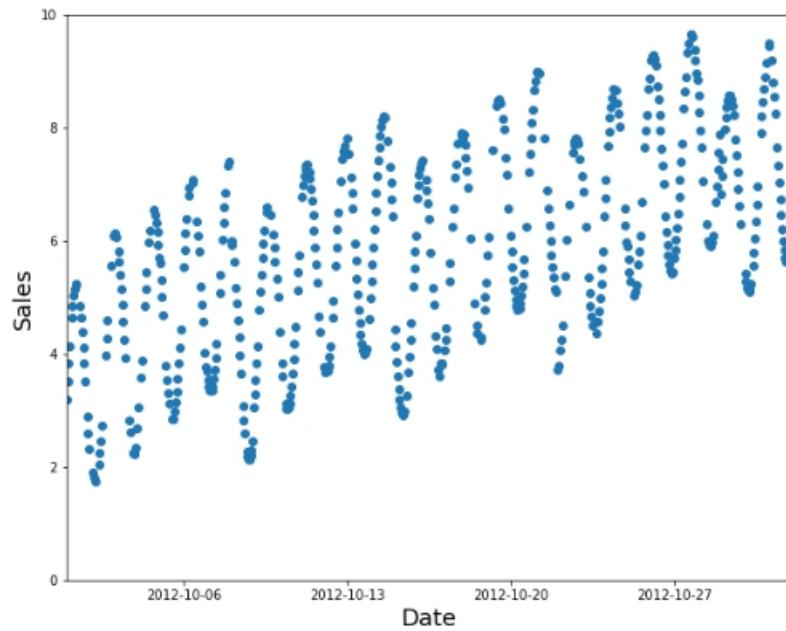
Suppose you want to split the time series data in <> into training and validation sets.

<>内の時系列データをトレーニングセットとバリデーションセットに分けたいとします。



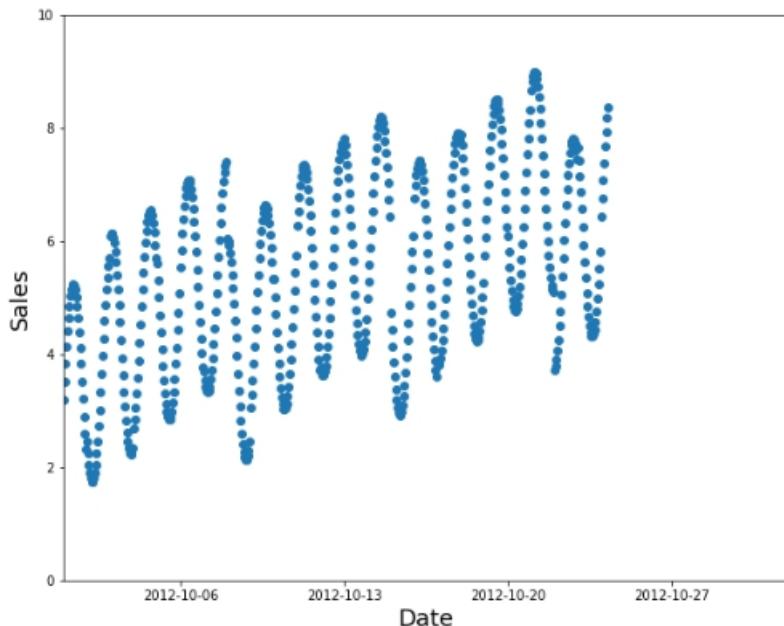
A random subset is a poor choice (too easy to fill in the gaps, and not indicative of what you'll need in production), as we can see in <>.

ランダムなサブセットは、<>でわかるように、良くない選択です(隙間を埋めるのが簡単すぎるし、本番で必要なものを示すものでもない)。



Instead, use the earlier data as your training set (and the later data for the validation set), as shown in <>.

その代わり、<>のように、先のデータをトレーニングセットとして(後のデータを検証セットとして)使用します。



For example, Kaggle had a competition to [predict the sales in a chain of Ecuadorian grocery stores](#). Kaggle's training data ran from Jan 1 2013 to Aug 15 2017, and the test data spanned Aug 16 2017 to Aug 31 2017. That way, the competition organizer ensured that entrants were making predictions for a time period that was *in the future*, from the perspective of their model. This is similar to the way quant hedge fund traders do *back-testing* to check whether their models are predictive of future periods, based on past data.

例えば、Kaggleでは、エクアドルの食料品店のチェーン店での売上を予測するコンペがありました。Kaggleのトレーニングデータは2013年1月1日から2017年8月15日までで、テストデータは2017年8月16日から2017年8月31日まででした。そうすることで、コンペティションの主催者は、応募者が自分のモデルの観点から、未来にある期間の予測を行っていることを確認したのです。これは、クオンツ・ヘッジファンドのトレーダーが、過去のデータに基づいて、自分のモデルが将来の期間を予測しているかどうかを確認するバックテストの方法と似ています。

A second common case is when you can easily anticipate ways the data you will be making predictions for in production may be *qualitatively different* from the data you have to train your model with.

2つ目によくあるケースは、本番で予測を行うデータが、モデルを訓練するためのデータとは質的に異なることが容易に予想される場合です。

In the Kaggle [distracted driver competition](#), the independent variables are pictures of drivers at the wheel of a car, and the dependent variables are categories such as texting, eating, or safely looking ahead. Lots of pictures are of the same drivers in different positions, as we can see in <<img_driver>>. If you were an insurance company building a model from this data, note that you would be most interested in how the model performs on drivers it hasn't seen before (since you would likely have training data only for a small group of people). In recognition of this, the test data for the competition consists of images of people that don't appear in the training set.

Kaggle の distracted driver コンペティションでは、独立変数は車のハンドルを握っているドライバーの写真で、従属変数はテキスト、食事、安全な前方確認などのカテゴリです。<>に見られるように、多くの写真は同じドライバーが異なる姿勢で写っているものです。もしあなたが保険会社でこのデータからモデルを作るとしたら、そのモデルが見たことのないドライバーに対してどのように機能するかに最も興味があるはずです（トレーニングデータは少人数のグループしか持っていないでしょうから）。そのため、このコンペティションのテストデータは、トレーニングセットにはない人物の画像で構成されています。



If you put one of the images in <> in your training set and one in the validation set, your model will have an easy time making a prediction for the one in the validation set, so it will seem to be performing better than it would on new people. Another perspective is that if you used all the people in training your model, your model might be overfitting to particularities of those specific people, and not just learning the states (texting, eating, etc.).

もし、<>の画像の 1 つをトレーニングセットに入れ、1 つを検証セットに入れた場合、モデルは検証セットの画像に対して簡単に予測を行うことができ、新しい人に対して行うよりも良いパフォーマンスを発揮するように見えます。もう一つの視点は、モデルのトレーニングにすべての人を使った場合、モデルは特定の人の特殊性にオーバーフィットしてしまい、単に状態（テキスト、食事など）を学習していないかもしれないということです。

A similar dynamic was at work in the [Kaggle fisheries competition](#) to identify the species of fish caught by fishing boats in order to reduce illegal fishing of endangered populations. The test set consisted of boats that didn't appear in the training data. This means that you'd want your validation set to include boats that are not in the training set.

同様の動きは、絶滅危惧種の違法漁業を減らすために、漁船が捕獲した魚の種類を特定する Kaggle の漁業コンペティションでも見受けられました。テストセットは、トレーニングデータには登場しない船で構成されていました。つまり、検証セットには、トレーニングセットにはないボートを含める必要があるということです。

Sometimes it may not be clear how your validation data will differ. For instance, for a problem using satellite

imagery, you'd need to gather more information on whether the training set just contained certain geographic locations, or if it came from geographically scattered data.

検証データがどのように異なるのかが明確でない場合もあります。例えば、衛星画像を使用する問題では、トレーニングセットに特定の地理的な場所が含まれているだけなのか、それとも地理的に散在したデータから得られたものなのか、さらに情報を収集する必要があります。

Now that you have gotten a taste of how to build a model, you can decide what you want to dig into next.

これで、モデルの作り方を一通り理解したところで、次は何を調べるかを決めればいい。

A *Choose Your Own Adventure* moment

自分で選ぶ冒険の瞬間

If you would like to learn more about how to use deep learning models in practice, including how to identify and fix errors, create a real working web application, and avoid your model causing unexpected harm to your organization or society more generally, then keep reading the next two chapters. If you would like to start learning the foundations of how deep learning works under the hood, skip to <>. (Did you ever read *Choose Your Own Adventure* books as a kid? Well, this is kind of like that... except with more deep learning than that book series contained.)

エラーの特定と修正、実際に動作するウェブアプリケーションの作成、あなたの組織や社会により一般的に予期せぬ害を及ぼすモデルの回避など、深層学習モデルを実際に使用する方法についてもっと学びたい場合は、次の 2 章を読み続けてください。深層学習がどのように機能するかの基礎を学びたい場合は、<>まで読み飛ばしてください。(子供のころに「冒険の旅」を読んだことがありますか？これはそのようなものです…ただし、その本シリーズに含まれている以上の深層学習が含まれていますが…)

You will need to read all these chapters to progress further in the book, but it is totally up to you which order you read them in. They don't depend on each other. If you skip ahead to <>, we will remind you at the end to come back and read the chapters you skipped over before you go any further.

この本をさらに進めるには、これらの章をすべて読む必要がありますが、どの順番で読むかは完全にあなた次第です。どの順番で読むかは自由です。もし、「」まで読み飛ばした場合は、最後に、読み飛ばした章をもう一度読んでから、それ以上進むように注意書きをします。

Questionnaire

アンケート

It can be hard to know in pages and pages of prose what the key things are that you really need to focus on and remember. So, we've prepared a list of questions and suggested steps to complete at the end of each chapter. All the answers are in the text of the chapter, so if you're not sure about anything here, reread that part of the text and make sure you understand it. Answers to all these questions are also available on the [book's website](#). You can also visit [the forums](#) if you get stuck to get help from other folks studying this material.

何ページにもわたる散文の中で、本当に注目し、覚えておかなければならぬ重要なことが何なのかを知るのは難しいかもしれません。そこで、各章の最後に、質問とその手順を提案するリストを用意しました。答えはすべてその章の本文に書いてありますので、ここでわからぬことがあれば、その部分を読み直して、確実に理解するようにしてください。また、これらすべての質問の答えは、この本のウェブサイトにも掲載されています。また、行き詰ったときはフォーラムを訪れ、この教材を勉強している他の人たちから助けを得ることもできます。

For more questions, including detailed answers and links to the video timeline, have a look at Radek Osmulski's [aiquizzes](#).

詳しい解答や動画のタイムラインへのリンクを含む、より多くの質問については、Radek Osmulski の aiquizzes を見てください。

1.

Do you need these for deep learning?

ディープラーニングに必要なものですか？

2.

- Lots of math T / F
- Lots of data T / F
- Lots of expensive computers T / F

高価なコンピュータがたくさんある

- A PhD T / F

3.

Name five areas where deep learning is now the best in the world.

ディープラーニングが今、世界で一番優れている分野を 5 つ挙げてください。

4.

5.

What was the name of the first device that was based on the principle of the artificial neuron?.

人工ニューロンの原理を応用した最初のデバイスの名前を教えてください。

6.

7.

Based on the book of the same name, what are the requirements for parallel distributed processing (PDP)?

同名の書籍に基づき、並列分散処理（PDP）に必要な要件は何か？

8.

9.

What were the two theoretical misunderstandings that held back the field of neural networks?

ニューラルネットワークの分野を阻んだ 2 つの理論的誤解とは何か？

10.

11.

What is a GPU?

GPU とは何か？

12.

13.

Open a notebook and execute a cell containing: $1+1$. What happens?

ノートブックを開き、 $1+1$ を含むセルを実行します。何が起こるか？

14.

15.

Follow through each cell of the stripped version of the notebook for this chapter. Before executing each cell, guess what will happen.

この章のノートブックの剥がされたバージョンの各セルを実行しなさい。各セルを実行する前に、何が起こるか推測してください。

16.

17.

Complete the Jupyter Notebook online appendix.

Jupyter Notebook のオンライン付録を完成させる。

18.

19.

Why is it hard to use a traditional computer program to recognize images in a photo?

従来のコンピュータプログラムを使って写真の画像を認識することが難しいのはなぜか？

20.

21.

What did Samuel mean by "weight assignment"?

サミュエルが言った「ウェイトアサイン」とはどういう意味か？

22.

23.

What term do we normally use in deep learning for what Samuel called "weights"?

サミュエルが「重み」と呼んだものを、深層学習では通常どのような言葉で表現しているのか？

24.

25.

Draw a picture that summarizes Samuel's view of a machine learning model.

サミュエルの考える機械学習モデルを要約した絵を描け。

26.

27.

Why is it hard to understand why a deep learning model makes a particular prediction?

ディープラーニングモデルが特定の予測をする理由を理解するのが難しいのはなぜか？

28.

29.

What is the name of the theorem that shows that a neural network can solve any mathematical problem to any level of accuracy?

ニューラルネットワークがあらゆる数学的問題をあらゆるレベルの精度で解くことができるることを示す定理の名前は？

30.

31.

What do you need in order to train a model?

モデルを訓練するために必要なものは何ですか？

32.

33.

How could a feedback loop impact the rollout of a predictive policing model?

フィードバックループは、予測警察モデルの展開にどのような影響を与えるか？

34.

35.

Do we always have to use 224×224-pixel images with the cat recognition model?

猫認識モデルでは、常に 224×224 ピクセルの画像を使用しなければならないのでしょうか？

36.

37.

What is the difference between classification and regression?

分類と回帰の違いは何ですか？

38.

39.

What is a validation set? What is a test set? Why do we need them?

バリデーションセットとは？ テストセットとは何ですか？ なぜそれらが必要なのか？

40.

41.

What will fastai do if you don't provide a validation set?

検証セットを提供しない場合、fastai は何をするのか？

42.

43.

Can we always use a random sample for a validation set? Why or why not?

検証セットには常にランダムなサンプルを使用することができますか？ なぜでしょうか、なぜでしょく？

44.

45.

What is overfitting? Provide an example.

オーバーフィッティングとは何か？ 例を挙げて説明しなさい。

46.

47.

What is a metric? How does it differ from "loss"?

メトリックとは何ですか？ 損失」とどう違うのですか？

48.

49.

How can pretrained models help?

事前学習したモデルはどのように役立つか？

50.

51.

What is the "head" of a model?

モデルの "頭" とは？

52.

53.

What kinds of features do the early layers of a CNN find? How about the later layers?

CNN の初期のレイヤーはどのような特徴を見出すか？後の層はどうだろう？

54.

55.

Are image models only useful for photos?

画像モデルは写真にしか使えないのか？

56.

57.

What is an "architecture"?

建築」ってなんですか？

58.

59.

What is segmentation?

セグメンテーションって何？

60.

61.

What is y_range used for? When do we need it?

y_range は何に使うのか？どのような時に必要なのか？

62.

63.

What are "hyperparameters"?

ハイパーパラメーター」とは何ですか？

64.

65.

What's the best way to avoid failures when using AI in an organization?

組織で AI を使うときに失敗しないための工夫とは？

66.

Further Research

さらなる研究

Each chapter also has a "Further Research" section that poses questions that aren't fully answered in the text, or gives more advanced assignments. Answers to these questions aren't on the book's website; you'll need to do your own research!

また、各章には「Further Research」という項目があり、本文では答えきれない疑問や、より高度な課題を提示しています。これらの質問に対する回答は、本書のウェブサイトには掲載されていませんので、ご自身で調べていただく必要があります！

1. Why is a GPU useful for deep learning? How is a CPU different, and why is it less effective for deep learning?

なぜ GPU がディープラーニングに有効なのか？CPU はどう違うのか、なぜディープラーニングには効果が薄いのか？

2. Try to think of three areas where feedback loops might impact the use of machine learning. See if you can find documented examples of that happening in practice.

フィードバックループが機械学習の使用に影響を与える可能性がある分野を 3 つ考えてみてください。実際に起こっている文書化された例を見つけることができるかどうか確認してください。