preface

The most significant event in the recent history of technology is perhaps the explosion in the power of neural networks since 2012. This was when the growth in labeled datasets, increases in computation power, and innovations in algorithms came together and reached a critical mass. Since then, deep neural networks have made previously unachievable tasks achievable and boosted the accuracies in other tasks, pushing them beyond academic research and into practical applications in domains such as speech recognition, image labeling, generative models, and recommendation systems, just to name a few.

It was against this backdrop that our team at Google Brain started developing TensorFlow.js. When the project started, many regarded "deep learning in JavaScript" as a novelty, perhaps a gimmick, fun for certain use cases, but not to be pursued with seriousness. While Python already had several well-established and powerful frameworks for deep learning, the JavaScript machine-learning landscape remained splintered and incomplete. Of the handful of JavaScript libraries available back then, most only supported deploying models pretrained in other languages (usually in Python). For the few that supported building and training models from scratch, the scope of supported model types was limited. Considering JavaScript's popular status and its ubiquity that straddles client and server sides, this was a strange situation.

TensorFlow.js is the first full-fledged industry-quality library for doing neural networks in JavaScript. The range of capabilities it provides spans multiple dimensions. First, it supports a wide range of neural-networks layers, suitable for various data types ranging from numeric to text, from audio to images. Second, it provides APIs for loading pretrained models for inference, fine-tuning pretrained models, and building and training models from scratch. Third, it provides both a high-level, Keras-like API for practitioners who opt to use well-established layer types, and a low-level, TensorFlow-like API for those who wish to implement more novel algorithms. Finally, it is designed

to be runnable in a wide selection of environments and hardware types, including the web browser, server side (Node.js), mobile (e.g., React Native and WeChat), and desktop (electron). Adding to the multidimensional capability of TensorFlow.js is its status as a first-class integrated part of the larger TensorFlow/Keras ecosystem, specifically its API consistency and two-way model-format compatibility with the Python libraries.

The book you have in your hands will guide your grand tour through this multidimensional space of capabilities. We've chosen a path that primarily cuts through the first dimension (modeling tasks), enriched by excursions along the remaining dimensions. We start from the relatively simpler task of predicting numbers from numbers (regression) to the more complex ones such as predicting classes from images and sequences, ending our trip on the fascinating topics of using neural networks to generate new images and training agents to make decisions (reinforcement learning).

We wrote the book not just as a recipe for how to write code in TensorFlow.js, but as an introductory course in the foundations of machine learning in the native language of JavaScript and web developers. The field of deep learning is a fast-evolving one. It is our belief that a firm understanding of machine learning is possible without formal mathematical treatment, and this understanding will enable you to keep yourself up-to-date in future evolution of the techniques.

With this book you've made the first step in becoming a member of the growing community of JavaScript machine-learning practitioners, who've already brought about many impactful applications at the intersection between JavaScript and deep learning. It is our sincere hope that this book will kindle your own creativity and ingenuity in this space.

SHANQING CAI, STAN BILESCHI, AND ERIC NIELSEN
September 2019
Cambridge, MA

1 Deep learning and JavaScript

This chapter covers

- What deep learning is and how it is related to artificial intelligence (AI) and machine learning
- What makes deep learning stand out among various machine-learning techniques, and the factors that led to the current "deep-learning revolution"
- The reasons for doing deep learning in JavaScript using TensorFlow.js
- The overall organization of this book

All the buzz around artificial intelligence (AI) is happening for a good reason: the deep-learning revolution, as it is sometimes called, has indeed happened. *Deep-learning revolution* refers to the rapid progress made in the speed and techniques of deep neural networks that started around 2012 and is still ongoing. Since then, deep neural networks have been applied to an increasingly wide range of problems, enabling machines to solve previously unsolvable problems in some cases and dramatically improving solution accuracy in others (see table 1.1 for examples). To experts in AI, many of these breakthroughs in neural networks were stunning.

To engineers who use neural networks, the opportunities this progress has created are galvanizing.

JavaScript is a language traditionally devoted to creating web browser UI and back-end business logic (with Node.js). As someone who expresses ideas and creativity in JavaScript, you may feel a little left out by the deep-learning revolution, which seems to be the exclusive territory of languages such as Python, R, and C++. This book aims at bringing deep learning and JavaScript together through the JavaScript deep-learning library called TensorFlow.js. We do this so that JavaScript developers like you can learn how to write deep neural networks without learning a new language; more importantly, we believe deep learning and JavaScript belong together.

The cross-pollination will create unique opportunities, ones unavailable in any other programming language. It goes both ways for JavaScript and deep learning. With JavaScript, deep-learning applications can run on more platforms, reach a wider audience, and become more visual and interactive. With deep learning, JavaScript developers can make their web apps more intelligent. We will describe how later in this chapter.

Table 1.1 lists some of the most exciting achievements of deep learning that we've seen in this deep-learning revolution so far. In this book, we have selected a number of these applications and created examples of how to implement them in TensorFlow.js, either in their full glory or in reduced form. These examples will be covered in depth in the coming chapters. Therefore, you will not stop at marveling at the breakthroughs: you can learn about them, understand them, and implement them all in JavaScript.

But before you dive into these exciting, hands-on deep-learning examples, we need to introduce the essential context around AI, deep learning, and neural networks.

Table 1.1 Examples of tasks in which accuracy improved significantly thanks to deep-learning techniques since the beginning of the deep-learning revolution around 2012. This list is by no means comprehensive. The pace of progress will undoubtedly continue in the coming months and years.

Machine-learning task	Representative deep-learning technology	Where we use TensorFlow.js to perform a similar task in this book
Categorizing the content of images	Deep convolutional neural networks (convnets) such as ResNet[a] and Inception[b] reduced the error rate in the ImageNet classification task from ~25% in 2011 to below 5% in 2017.[c]	Training convnets for MNIST (chapter 4); MobileNet inference and transfer learning (chapter 5)

a. Kaiming He et al., "Deep Residual Learning for Image Recognition," *Proc. IEEE Conference Computer Vision and Pattern Recognition* (CVPR), 2016, pp. 770–778, http://mng.bz/PO5P.
b. Christian Szegedy et al., "Going Deeper with Convolutions," *Proc. IEEE Conference Computer Vision and Pattern Recognition* (CVPR), 2015, pp. 1–9, http://mng.bz/JzGv.
c. Large Scale Visual Recognition Challenge 2017 (ILSVRC2017) results, http://image-net.org/challenges/LSVRC/2017/results.

Table 1.1 Examples of tasks in which accuracy improved significantly thanks to deep-learning techniques since the beginning of the deep-learning revolution around 2012. This list is by no means comprehensive. The pace of progress will undoubtedly continue in the coming months and years. *(continued)*

Machine-learning task	Representative deep-learning technology	Where we use TensorFlow.js to perform a similar task in this book
Localizing objects and images	Variants of deep convnets[d] reduced localization error from 0.33 in 2012 to 0.06 in 2017.	YOLO in TensorFlow.js (section 5.2)
Translating one natural language to another	Google's neural machine translation (GNMT) reduced translation error by ~60% compared to the best traditional machine-translation techniques.[e]	Long Short-Term Memory (LSTM)-based sequence-to-sequence models with attention mechanisms (chapter 9)
Recognizing large-vocabulary, continuous speech	An LSTM-based encoder-attention-decoder architecture achieves a lower word-error rate than the best non-deep-learning speech recognition system.[f]	Attention-based LSTM small-vocabulary continuous speech recognition (chapter 9)
Generating realistic-looking images	Generative adversarial networks (GANs) are now capable of generating realistic-looking images based on training data (see https://github.com/junyanz/CycleGAN).	Generating images using variational autoencoders (VAEs) and GANs (chapter 9)
Generating music	Recurrent neural networks (RNNs) and VAEs are helping create music scores and novel instrument sounds (see https://magenta.tensorflow.org/demos).	Training LSTMs to generate text (chapter 9)
Learning to play games	Deep learning combined with reinforcement learning (RL) lets machines learn to play simple Atari games using raw pixels as the only input.[g] Combining deep learning and Monte Carlo tree search, AlphaZero reached a super-human level of Go purely through self-play.[h]	Using RL to solve the cart-pole control problem and a snake video game (chapter 11)
Diagnosing diseases using medical images	Deep convnets were able to achieve specificity and sensitivity comparable to trained human ophthalmologists in diagnosing diabetic retinopathy based on images of patients' retinas.[i]	Transfer learning using a pre-trained MobileNet image model (chapter 5).

d. Yunpeng Chen et al., "Dual Path Networks," https://arxiv.org/pdf/1707.01629.pdf.
e. Yonghui Wu et al., "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," submitted 26 Sept. 2016, https://arxiv.org/abs/1609.08144.
f. Chung-Cheng Chiu et al., "State-of-the-Art Speech Recognition with Sequence-to-Sequence Models," submitted 5 Dec. 2017, https://arxiv.org/abs/1712.01769.
g. Volodymyr Mnih et al., "Playing Atari with Deep Reinforcement Learning," NIPS Deep Learning Workshop 2013, https://arxiv.org/abs/1312.5602.
h. David Silver et al., "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," submitted 5 Dec. 2017, https://arxiv.org/abs/1712.01815.
i. Varun Gulshan et al., "Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs," JAMA, vol. 316, no. 22, 2016, pp. 2402–2410, http://mng.bz/wlDQ.

1.1 Artificial intelligence, machine learning, neural networks, and deep learning

Phrases like *AI*, *machine learning*, *neural networks*, and *deep learning* mean related but different things. To orient yourself in the dazzling world of AI, you need to understand what they refer to. Let's define these terms and the relations among them.

1.1.1 Artificial intelligence

As the Venn diagram in figure 1.1 shows, AI is a broad field. A concise definition of the field would be as follows: *the effort to automate intellectual tasks normally performed by humans*. As such, AI encompasses machine learning, neural networks, and deep learning, but it also includes many approaches distinct from machine learning. Early chess programs, for instance, involved hard-coded rules crafted by programmers. Those didn't qualify as machine learning because the machines were programmed explicitly to solve the problems instead of being allowed to discover strategies for solving the problems by learning from the data. For a long time, many experts believed that

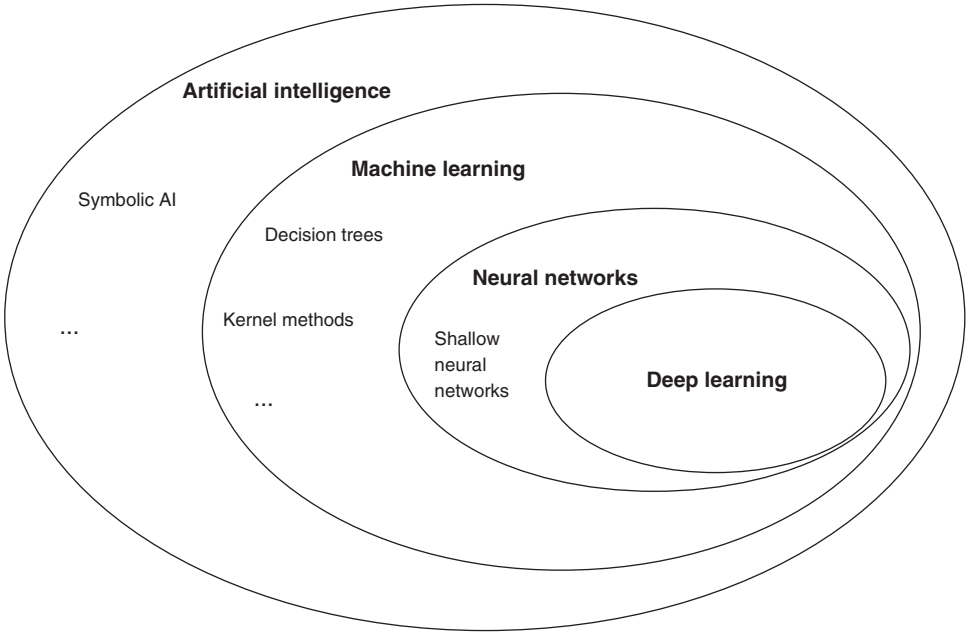

Figure 1.1 Relations between AI, machine learning, neural networks, and deep learning. As this Venn diagram shows, machine learning is a subfield of AI. Some areas of AI use approaches different from machine learning, such as symbolic AI. Neural networks are a subfield of machine learning. There exist non-neural-network machine-learning techniques, such as decision trees. Deep learning is the science and art of creating and applying "deep" neural networks—neural networks with multiple "layers"—versus "shallow" neural networks—neural networks with fewer layers.

human-level AI could be achieved through handcrafting a sufficiently large set of explicit rules for manipulating knowledge and making decisions. This approach is known as *symbolic AI*, and it was the dominant paradigm in AI from the 1950s to the late 1980s.[1]

1.1.2 Machine learning: How it differs from traditional programming

Machine learning, as a subfield of AI distinct from symbolic AI, arises from a question: Could a computer go beyond what a programmer knows how to program it to perform, and learn on its own how to perform a specific task? As you can see, the approach of machine learning is fundamentally different from that of symbolic AI. Whereas symbolic AI relies on hard-coding knowledge and rules, machine learning seeks to avoid this hard-coding. So, if a machine isn't explicitly instructed on how to perform a task, how would it learn how to do so? The answer is by learning from examples in the data.

This opened the door to a new programming paradigm (figure 1.2). To give an example of the machine-learning paradigm, let's suppose you are working on a web app that handles photos uploaded by users. A feature you want in the app is automatic classification of photos into ones that contain human faces and ones that don't. The app will take different actions on face images and no-face images. To this end, you want to create a program to output a binary face/no-face answer given any input image (made of an array of pixels).

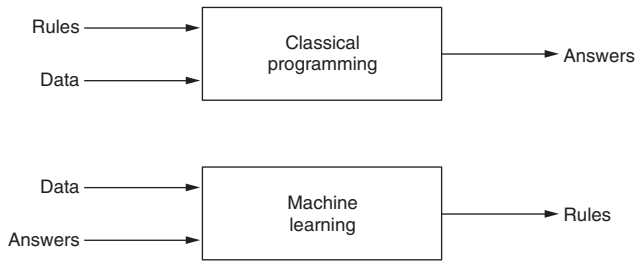

Figure 1.2 Comparing the classical programming paradigm and the machine-learning paradigm

We humans can perform this task in a split second: our brains' genetic hardwiring and life experience give us the ability to do so. However, it is hard for any programmer, no matter how smart and experienced, to write an explicit set of rules in a programming language (the only practical way for humans to communicate with a computer) on how to accurately decide whether an image contains a human face. You can spend days poring over code that does arithmetic on the RGB (red-green-blue) values of pixels to detect elliptic contours that look like faces, eyes, and mouths, as well as devising heuristic rules on the geometric relations between the contours. But you will soon realize that such effort is laden with arbitrary choices of logic and parameters that are

[1] An important type of symbolic AI is *expert systems*. See this Britannica article to learn about them: http://mng.bz/7zmy.

hard to justify. More importantly, it is hard to make it work well![2] Any heuristic you come up with is likely to fall short when facing the myriad variations that faces can present in real-life images, such as differences in the size, shape, and details of the face; facial expression; hairstyle; skin color; orientation; the presence or absence of partial obscuring; glasses; lighting conditions; objects in the background; and so on.

In the machine-learning paradigm, you recognize that handcrafting a set of rules for such a task is futile. Instead, you find a set of images, some with faces in them and some without. Then you enter the desired (that is, correct) face or no-face answer for each one. These answers are referred to as *labels*. This is a much more tractable (in fact, trivial) task. It may take some time to label all the images if there are a lot of them, but the labeling task can be divided among several humans and can proceed in parallel. Once you have the images labeled, you apply machine learning and let machines discover the set of rules on their own. If you use the correct machine-learning techniques, you will arrive at a trained set of rules capable of performing the face/no-face task with an accuracy > 99%—far better than anything you can hope to achieve with handcrafted rules.

From the previous example, we can see that machine learning is the process of automating the discovery of rules for solving complex problems. This automation is beneficial for problems like face detection, in which humans know the rules intuitively and can easily label the data. For other problems, the rules are not known intuitively. For example, consider the problem of predicting whether a user will click an ad displayed on a web page, given the page's and the ad's contents and other information, such as time and location. No human has a good sense about how to make accurate predictions for such problems in general. Even if one does, the pattern will probably change with time and with the appearance of new content and new ads. But the labeled training data is available from the ad service's history: it is available from the ad servers' logs. The availability of the data and labels alone makes machine learning a good fit for problems like this.

In figure 1.3, we take a closer look at the steps involved in machine learning. There are two important phases. The first is the *training phase*. This phase takes the data and answers, together referred to as the *training data*. Each pair of input data and the desired answer is called an *example*. With the help of the examples, the training process produces the automatically discovered *rules*. Although the rules are discovered automatically, they are not discovered entirely from scratch. In other words, machine-learning algorithms are not creative in coming up with rules. In particular, a human engineer provides a blueprint for the rules at the outset of training. The blueprint is encapsulated in a *model*, which forms a *hypothesis space* for the rules the machine may possibly learn. Without this hypothesis space, there is a completely unconstrained and infinite space of possible rules to search in, which is not conducive to finding good

[2] In fact, such approaches have indeed been attempted before and did not work very well. This survey paper provides good examples of handcrafting rules for face detection before the advent of deep learning: Erik Hjelmås and Boon Kee Low, "Face Detection: A Survey," *Computer Vision and Image Understanding*, Sept. 2001, pp. 236–274, http://mng.bz/m4d2.

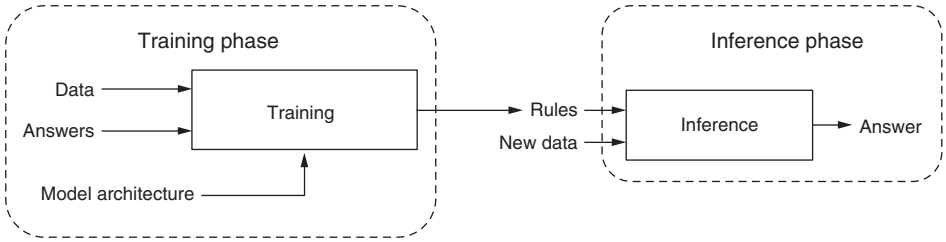

Figure 1.3 A more detailed view of the machine-learning paradigm than that in figure 1.2. The workflow of machine learning consists of two phases: training and inference. Training is the process of the machine automatically discovering the rules that convert the data into answers. The learned rules, encapsulated in a trained "model," are the fruit of the training phase and form the basis of the inference phase. Inference means using the model to obtain answers for new data.

rules in a limited amount of time. We will describe in great detail the kinds of models available and how to choose the best ones based on the problem at hand. For now, it suffices to say that in the context of deep learning, models vary in terms of how many layers the neural network consists of, what types of layers they are, and how they are wired together.

With the training data and the model architecture, the training process produces the learned rules, encapsulated in a trained model. This process takes the blueprint and alters (or tunes) it in ways that nudge the model's output closer and closer to the desired output. The training phase can take anywhere from milliseconds to days, depending on the amount of training data, the complexity of the model architecture, and how fast the hardware is. This style of machine learning—namely, using labeled examples to progressively reduce the error in a model's outputs—is known as *supervised learning*.[3] Most of the deep-learning algorithms we cover in this book are supervised learning. Once we have the trained model, we are ready to apply the learned rules on new data—data that the training process has never seen. This is the second phase, or *inference phase*. The inference phase is less computationally intensive than the training phase because 1) inference usually happens on one input (for instance, one image) at a time, whereas training involves going through all the training data; and 2) during inference, the model does not need to be altered.

LEARNING REPRESENTATIONS OF DATA

Machine learning is about learning from data. But *what* exactly is learned? The answer: a way to effectively transform the data or, in other words, to change the old representations of the data into a new one that gets us closer to solving the problem at hand.

[3] Another style of machine learning is *unsupervised learning*, in which unlabeled data is used. Examples of unsupervised learning are clustering (discovering distinct subsets of examples in a dataset) and anomaly detection (determining if a given example is sufficiently different from the examples in the training set).

Before we go any further, what is a representation? At its core, it is a way to look at the data. The same data can be looked at in different ways, leading to different representations. For example, a color image can have an RGB or HSV (hue-saturation-value) encoding. Here, the words *encoding* and *representation* mean essentially the same thing and can be used interchangeably. When encoded in these two different formats, the numerical values that represent the pixels are completely different, even though they are for the same image. Different representations are useful for solving different problems. For example, to find all the red parts of an image, the RGB representation is more useful; but to find color-saturated parts of the same image, the HSV representation is more useful. This is essentially what machine learning is all about: finding an appropriate transformation that turns the old representation of the input data into a new one—one that is amenable to solving the specific task at hand, such as detecting the location of cars in an image or deciding whether an image contains a cat and a dog.

To give a visual example, we have a collection of white points and several black points in a plane (figure 1.4). Let's say we want to develop an algorithm that can take the 2D (x, y) coordinates of a point and predict whether that point is black or white. In this case,

- The input data is the two-dimensional Cartesian coordinates (x and y) of a point.
- The output is the predicted color of the point (whether it's black or white).

The data shows a pattern in panel A of figure 1.4. How would the machine decide the color of a point given the x- and y-coordinates? It cannot simply compare x with a number, because the range of the x-coordinates of the white points overlaps with the range of the x-coordinates of the black ones! Similarly, the algorithm cannot rely on the y-coordinate. Therefore, we can see that the original representation of the points is not a good one for the black-white classification task.

What we need is a new representation that separates the two colors in a more straightforward way. Here, we transform the original Cartesian x-y representation into a polar-coordinate-system representation. In other words, we represent a point by 1) its angle—the angle formed by the x-axis and the line that connects the origin with the point (see the example in panel A of figure 1.4) and 2) its radius—its distance from the origin. After this transformation, we arrive at a new representation of the same set of data, as panel B of figure 1.4 shows. This representation is more amenable to our task, in that the angle values of the black and white points are now completely nonoverlapping. However, this new representation is still not an ideal one in that the black-white color classification cannot be made into a simple comparison with a threshold value (like zero).

Luckily, we can apply a second transformation to get us there. This transformation is based on the simple formula

```
(absolute value of angle) - 135 degrees
```