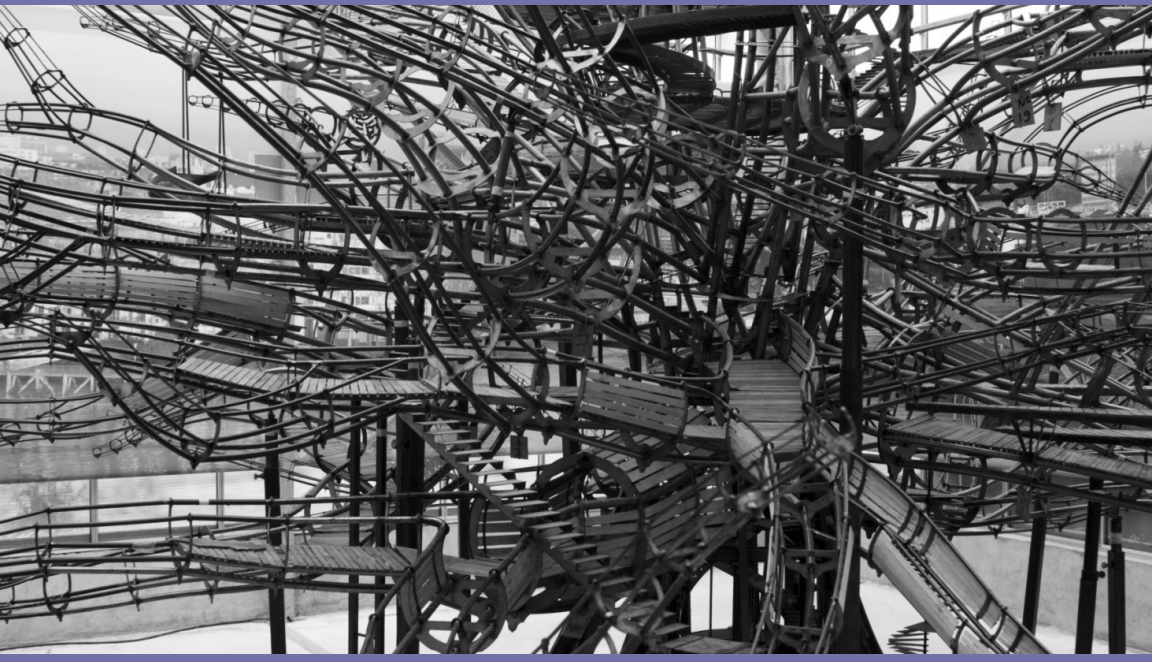
O'REILLY®

Machine Learning for Designers



Patrick Hebron

Short. Smart. Seriously useful.

Free ebooks and reports from O'Reilly at **oreil.ly/fr-design**

Free ebooks, reports and other articles on UX design, data-informed design, and design for the IoT. Get insights from industry experts and stay current with the latest developments from O'Reilly.

9 781491 956205

O'REILLY®

Short. Smart. Seriously useful.

Free ebooks and reports from O'Reilly at **oreil.ly/fr-design**

Free ebooks, reports and other articles on UX design, data-informed design, and design for the IoT. Get insights from industry experts and stay current with the latest developments from O'Reilly.

O'REILLY®

Machine Learning for Designers

Patrick Hebron

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Machine Learning for Designers
by Patrick Hebron

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Editor: Angela Rufino
Production Editor: Shiny Kalapurakkel
Copyeditor: Dianne Russell, Octal Publishing, Inc.
Proofreader: Molly Ives Brower
Interior Designer: David Futato
Cover Designer: Randy Comer
Illustrator: Rebecca Panzer

June 2016: First Edition

Revision History for the First Edition
2016-06-09: First Release

978-1-491-95620-5

[LSI]

Table of Contents

Machine Learning for Designers

Introduction

Since the dawn of computing, we have dreamed of (and had nightmares about) machines that can think and speak like us. But the computers we've interacted with over the past few decades are a far cry from HAL 9000 or Samantha from *Her*. Nevertheless, machine learning is in the midst of a renaissance that will transform countless industries and provide designers with a wide assortment of new tools for better engaging with and understanding users. These technologies will give rise to new design challenges and require new ways of thinking about the design of user interfaces and interactions.

To take full advantage of these systems' vast technical capabilities, designers will need to forge even deeper collaborative relationships with programmers. As these complex technologies make their way from research prototypes to user-facing products, programmers will also rely upon designers to discover engaging applications for these systems.

In the text that follows, we will explore some of the technical properties and constraints of machine learning systems as well as their implications for user-facing designs. We will look at how designers can develop interaction paradigms and a design vocabulary around these technologies and consider how designers can begin to incorporate the power of machine learning into their work.

Why Design for Machine Learning is Different

A Different Kind of Logic

In our everyday communication, we generally use what logicians call *fuzzy logic*. This form of logic relates to approximate rather than exact reasoning. For example, we might identify an object as being "very small," "slightly red," or "pretty nearby." These statements do not hold an exact meaning and are often context-dependent. When we say that a car is small, this implies a very different scale than when we say that a planet is small. Describing an object in these terms requires an auxiliary knowledge of the range of possible values that exists within a specific domain of meaning. If we had only seen one car ever, we would not be able to distinguish a small car from a large one. Even if we had seen a handful of cars, we could not say with great assurance that we knew the full range of possible car sizes. With sufficient experience, we could never be completely sure that we had seen the smallest and largest of all cars, but we could feel relatively certain that we had a good approximation of the range. Since the people around us will tend to have had relatively similar experiences of cars, we can meaningfully discuss them with one another in fuzzy terms.

Computers, however, have not traditionally had access to this sort of auxiliary knowledge. Instead, they have lived a life of experiential deprivation. As such, traditional computing platforms have been designed to operate on logical expressions that can be evaluated without the knowledge of any outside factor beyond those expressly provided to them. Though fuzzy logical expressions can be employed by traditional platforms through the programmer's or user's explicit delineation of a fuzzy term such as "very small," these systems have generally been designed to deal with *boolean logic* (also called "binary logic"), in which every expression must ultimately evaluate to either true or false. One rationale for this approach, as we will discuss further in the next section, is that boolean logic allows a computer program's behavior to be defined as a finite set of concrete states, making it easier to build and test systems that will behave in a predictable manner and conform precisely to their programmer's intentions.

Machine learning changes all this by providing mechanisms for imparting experiential knowledge upon computing systems. These

technologies enable machines to deal with fuzzier and more complex or "human" concepts, but also bring an assortment of design challenges related to the sometimes problematic nature of working with imprecise terminology and unpredictable behavior.

A Different Kind of Development

In traditional programming environments, developers use boolean logic to explicitly describe each of a program's possible states and the exact conditions under which the user will be able to transition between them. This is analogous to a "choose-your-own-adventure" book, which contains instructions like, "if you want the prince to fight the dragon, turn to page 32." In code, a *conditional expression* (also called an *if-statement*) is employed to move the user to a particular portion of the code if some pre defined set of conditions is met.

In pseudocode, a conditional expression might look like this:

```
if ( mouse button is pressed and mouse is over the 'Login'
button ),
then show the 'Welcome' screen
```

Since a program comprises a finite number of states and transitions, which can be explicitly enumerated and inspected, the program's overall behavior should be predictable, repeatable, and testable. This is not to say, of course, that traditional programmatic logic cannot contain hard-to-foresee "edge-cases," which lead to undefined or undesirable behavior under some specific set of conditions that have not been addressed by the programmer. Yet, regardless of the difficulty of identifying these problematic edge-cases in a complex piece of software, it is at least conceptually possible to methodically probe every possible path within the "choose-your-own-adventure" and prevent the user from accessing an undesirable state by altering or appending the program's explicitly defined logic.

The behavior of machine learning systems, on the other hand, is not defined through this kind of explicit programming process. Instead of using an explicit set of rules to describe a program's possible behaviors, a machine learning system looks for patterns within a set of example behaviors in order to produce an approximate representation of the rules themselves.

This process is somewhat like our own mental processes for learning about the world around us. Long before we encounter any formal

description of the "laws" of physics, we learn to operate within them by observing the outcomes of our interactions with the physical world. A child may have no awareness of Newton's equations, but through repeated observation and experimentation, the child will come to recognize patterns in the relationships between the physical properties and behaviors of objects.

While this approach offers an extremely effective mechanism for learning to operate on complex systems, it does not yield a concrete or explicit set of rules governing that system. In the context of human intelligence, we often refer to this as "intuition," or the ability to operate on complex systems without being able to formally articulate the procedure by which we achieved some desired outcome. Informed by experience, we come up with a set of approximate or provisional rules known as *heuristics* (or "rules of thumb") and operate on that basis.

In a machine learning system, these implicitly defined rules look nothing like the explicitly defined logical expressions of a traditional programming language. Instead, they are comprised of distributed representations that implicitly describe the probabilistic connections between the set of interrelated components of a complex system.

Machine learning often requires a very large number of examples to produce a strong intuition for the behaviors of a complex system.

In a sense, this requirement is related to the problem of edge-cases, which present a different set of challenges in the context of machine learning. Just as it is hard to imagine every possible outcome of a set of rules, it is, conversely, difficult to extrapolate every possible rule from a set of example outcomes. To extrapolate a good approximation of the rules, the learner must observe many variations of their application. The learner must be exposed to the more extreme or unlikely behaviors of a system as well as the most likely ones. Or, as the educational philosopher Patricia Carini said, "To let meaning occur requires time and the possibility for the rich and varied relationships among things to become evident."[1]

While intuitive learners may be slower at rote procedural tasks such as those performed by a calculator, they are able to perform much more complex tasks that do not lend themselves to exact proce-

1 Patricia F. Carini, *On Value in Education* (New York, NY: Workshop Center, 1987).

dures. Nevertheless, even with an immense amount of training, these intuitive approaches sometimes fail us. We may, for instance, find ourselves mistakenly identifying a human face in a cloud or a grilled cheese sandwich.

A Different Kind of Precision

A key principle in the design of conventional programming languages is that each feature should work in a predictable, repeatable manner provided that the feature is being used correctly by the programmer. No matter how many times we perform an arithmetic operation such as "2 + 2," we should always get the same answer. If this is ever untrue, then a bug exists in the language or tool we are using. Though it is not inconceivable for a programming language to contain a bug, it is relatively rare and would almost never pertain to an operation as commonly used as an arithmetic operator. To be extra certain that conventional code will operate as expected, most large-scale codebases ship with a set of formal "unit tests" that can be run on the user's machine at installation time to ensure that the functionality of the system is fully in line with the developer's expectations.

So, putting rare bugs aside, conventional programming languages can be thought of as systems that are always correct about mundane things like concrete mathematical operations. Machine learning algorithms, on the other hand, can be thought of as systems that are often correct about more complicated things like identifying human faces in an image. Since a machine learning system is designed to probabilistically approximate a set of demonstrated behaviors, its very nature generally precludes it from behaving in an entirely predictable and reproducible manner, even if it has been properly trained on an extremely large number of examples. This is not to say, of course, that a well-trained machine learning system's behavior must inherently be erratic to a detrimental degree. Rather, it should be understood and considered within the design of machine-learning-enhanced systems that their capacity for dealing with extraordinarily complex concepts and patterns also comes with a certain degree of imprecision and unpredictability beyond what can be expected from traditional computing platforms.

Later in the text, we will take a closer look at some design strategies for dealing with imprecision and unpredictable behaviors in machine learning systems.

A Different Kind of Problem

Machine learning can perform complex tasks that cannot be addressed by conventional computing platforms. However, the process of training and utilizing machine learning systems often comes with substantially greater overhead than the process of developing conventional systems. So while machine learning systems can be taught to perform simple tasks such as arithmetic operations, as a general rule of thumb, you should only take a machine learning approach to a given problem if no viable conventional approach exists.

Even for tasks that are well-suited to a machine learning solution, there are numerous considerations about which learning mechanisms to use and how to curate the training data so that it can be most comprehensible to the learning system.

In the sections that follow, we will look more closely at how to identify problems that are well-suited for machine learning solutions as well as the numerous factors that go into applying learning algorithms to specific problems. But for the time being, we should understand machine learning to be useful in solving problems that can be encapsulated by a set of examples, but not easily described in formal terms.

What Is Machine Learning?

The Mental Process of Recognizing Objects

Think about your own mental process of recognizing a human face. It's such an innate, automatic behavior, it is difficult to think about in concrete terms. But this difficulty is not only a product of the fact that you have performed the task so many times. There are many other often-repeated procedures that we could express concretely, like how to brush your teeth or scramble an egg. Rather, it is nearly impossible to describe the process of recognizing a face because it involves the balancing of an extremely large and complex set of interrelated factors, and therefore defies any concrete description as a sequence of steps or set of rules.

To begin with, there is a great deal of variation in the facial features of people of different ethnicities, ages, and genders. Furthermore, every individual person can be viewed from an infinite number of

vantage points in countless lighting scenarios and surrounding environments. In assessing whether the object we are looking at is a human face, we must consider each of these properties in relation to each other. As we change vantage points around the face, the proportion and relative placement of the nose changes in relation to the eyes. As the face moves closer to or further from other objects and light sources, its coloring and regions of contrast change too.

There are infinite combinations of properties that would yield the valid identification of a human face and an equally great number of combinations that would not. The set of rules separating these two groups is just too complex to describe through conditional logic. We are able to identify a face almost automatically because our great wealth of experience in observing and interacting with the visible world has allowed us to build up a set of heuristics that can be used to quickly, intuitively, and somewhat imprecisely gauge whether a particular expression of properties is in the correct balance to form a human face.

Learning by Example

In logic, there are two main approaches to reasoning about how a set of specific observations and a set of general rules relate to one another. In *deductive reasoning*, we start with a broad theory about the rules governing a system, distill this theory into more specific hypotheses, gather specific observations and test them against our hypotheses in order to confirm whether the original theory was correct. In *inductive reasoning*, we start with a group of specific observations, look for patterns in those observations, formulate tentative hypotheses, and ultimately try to produce a general theory that encompasses our original observations. See Figure 1-1 for an illustration of the differences between these two forms of reasoning.

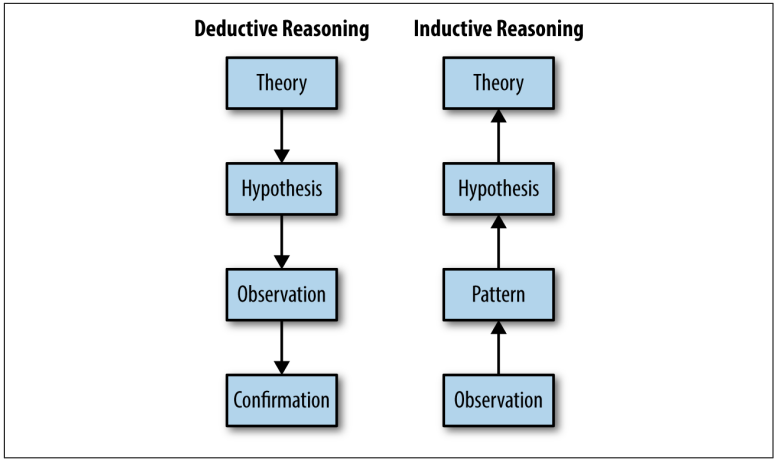


Figure 1-1. Deductive reasoning versus inductive reasoning

Each of these approaches plays an important role in scientific inquiry. In some cases, we have a general sense of the principles that govern a system, but need to confirm that our beliefs hold true across many specific instances. In other cases, we have made a set of observations and wish to develop a general theory that explains them.

To a large extent, machine learning systems can be seen as tools that assist or automate inductive reasoning processes. In a simple system that is governed by a small number of rules, it is often quite easy to produce a general theory from a handful of specific examples. Consider Figure 1-2 as an example of such a system.[2]

2 Zoltan P. Dienes and E. W. Golding, *Learning Logic, Logical Games* (Harlow [England] ESA, 1966).

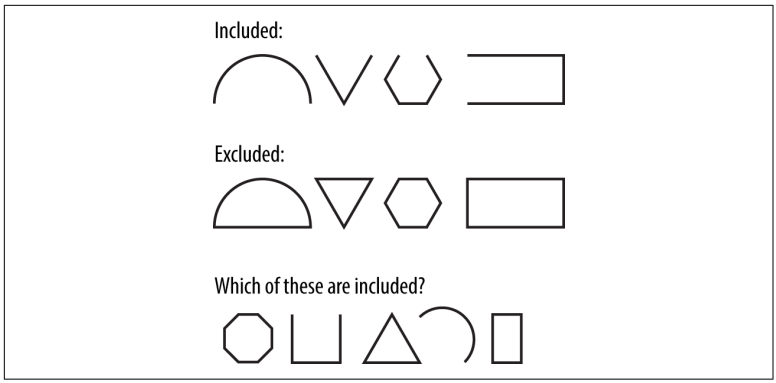


Figure 1-2. A simple system

In this system, you should have no trouble uncovering the singular rule that governs inclusion: open figures are included and closed figures are excluded. Once discovered, you can easily apply this rule to the uncategorized figures in the bottom row.

In Figure 1-3, you may have to look a bit harder.

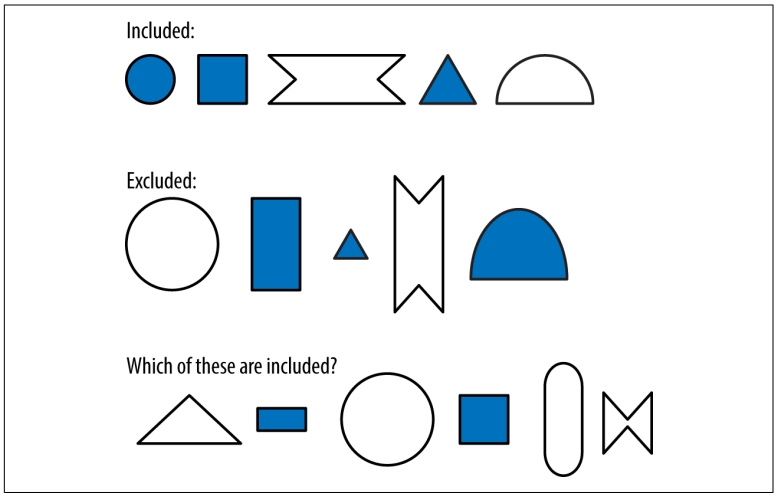


Figure 1-3. A more complex system

Here, there seem to be more variables involved. You may have considered the shape and shading of each figure before discovering that in fact this system is also governed by a single attribute: the figure's height. If it took you a moment to discover the rule, it is likely because you spent time considering attributes that seemed like they

would be pertinent to the determination but were ultimately not. This kind of "noise" exists in many systems, making it more difficult to isolate the meaningful attributes.

Let's now consider Figure 1-4.

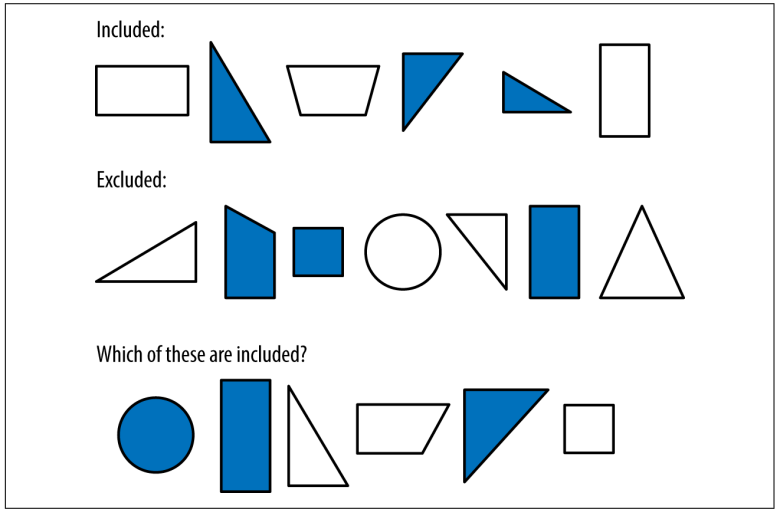


Figure 1-4. An even more complex system

In this diagram, the rules have in fact gotten a bit more complicated. Here, shaded triangles and unshaded quadrilaterals are included and all other figures are excluded. This rule system is harder to uncover because it involves an interdependency between two attributes of the figures. Neither the shape nor the shading alone determines inclusion. A triangle's inclusion depends upon its shading and a shaded figure's inclusion depends upon its shape. In machine learning, this is called a *linearly inseparable* problem because it is not possible to separate the included and excluded figures using a single "line" or determining attribute. Linearly inseparable problems are more difficult for machine learning systems to solve, and it took several decades of research to discover robust techniques for handling them. See Figure 1-5.

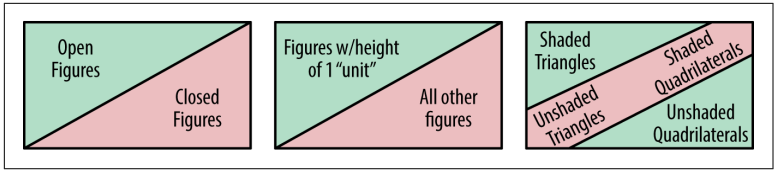


Figure 1-5. Linearly separable versus linearly inseparable problems

In general, the difficulty of an inductive reasoning problem relates to the number of relevant and irrelevant attributes involved as well as the subtlety and interdependency of the relevant attributes. Many real-world problems, like recognizing a human face, involve an immense number of interrelated attributes and a great deal of noise. For the vast majority of human history, this kind of problem has been beyond the reach of mechanical automation. The advent of machine learning and the ability to automate the synthesis of general knowledge about complex systems from specific information has deeply significant and far-reaching implications. For designers, it means being able to understand users more holistically through their interactions with the interfaces and experiences we build. This understanding will allow us to better anticipate and meet users' needs, elevate their capabilities and extend their reach.

Mechanical Induction

To get a better sense of how machine learning algorithms actually perform induction, let's consider Figure 1-6.

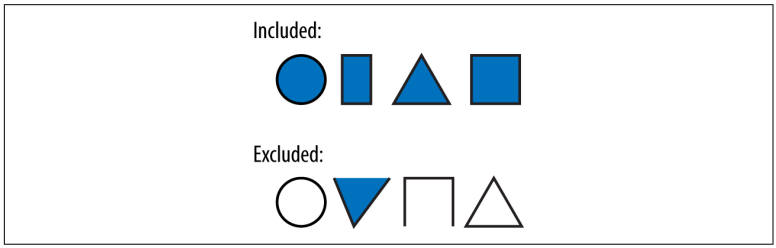


Figure 1-6. A system equivalent to the boolean logical expression, "AND"

This system is equivalent to the boolean logical expression, "AND." That is, only figures that are both shaded and closed are included. Before we turn our attention to induction, let's first consider how we would implement this logic in an electrical system from a deductive point of view. In other words, if we already knew the rule governing

this system, how could we implement an electrical device that determines whether a particular figure should be included or excluded? See Figure 1-7.

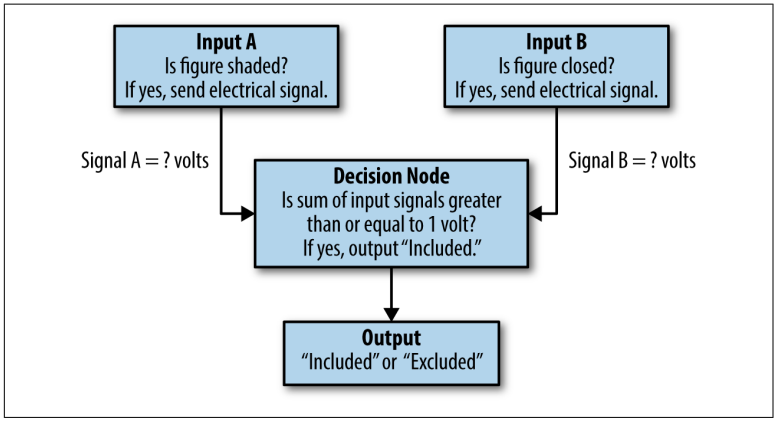


Figure 1-7. The boolean logical expression AND represented as an electrical circuit

In this diagram, we have a wire leading from each input attribute to a "decision node." If a given figure is shaded, then an electrical signal will be sent through the wire leading from Input A. If the figure is closed, then an electrical signal will be sent through the wire leading from Input B. The decision node will output an electrical signal indicating that the figure is included if the sum of its input signals is greater than or equal to 1 volt.

To implement the behavior of an AND gate, we need to set the voltage associated with each of the two input signals. Since the output threshold is 1 volt and we only want the output to be triggered if both inputs are active, we can set the voltage associated with each input to 0.5 volts. In this configuration, if only one or neither input is active, the output threshold will not be reached. With these signal voltages now set, we have implemented the mechanics of the general rule governing the system and can use this electronic device to deduce the correct output for any example input.

Now, let us consider the same problem from an inductive point of view. In this case, we have a set of example inputs and outputs that exemplify a rule but do not know what the rule is. We wish to determine the nature of the rule using these examples.

Let's again assume that the decision node's output threshold is 1 volt. To reproduce the behavior of the AND gate by induction, we need to find voltage levels for the input signals that will produce the expected output for each pair of example inputs, telling us whether those inputs are included in the rule. The process of discovering the right combination of voltages can be seen as a kind of search problem.

One approach we might take is to choose random voltages for the input signals, use these to predict the output of each example, and compare these predictions to the given outputs. If the predictions match the correct outputs, then we have found good voltage levels. If not, we could choose new random voltages and start the process over. This process could then be repeated until the voltages of each input were weighted so that the system could consistently predict whether each input pair fits the rule.

In a simple system like this one, a guess-and-check approach may allow us to arrive at suitable voltages within a reasonable amount of time. But for a system that involves many more attributes, the number of possible combinations of signal voltages would be immense and we would be unlikely to guess suitable values efficiently. With each additional attribute, we would need to search for a needle in an increasingly large haystack.

Rather than guessing randomly and starting over when the results are not suitable, we could instead take an iterative approach. We could start with random values and check the output predictions they yield. But rather than starting over if the results are inaccurate, we could instead look at the extent and direction of that inaccuracy and try to incrementally adjust the voltages to produce more accurate results. The process outlined above is a simplified description of the learning procedure used by one of the earliest machine learning systems, called a *Perceptron* (Figure 1-8), which was invented by Frank Rosenblatt in 1957.[3]

3 Frank Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," Psychological Review 65, no. 6 (1958): 386.

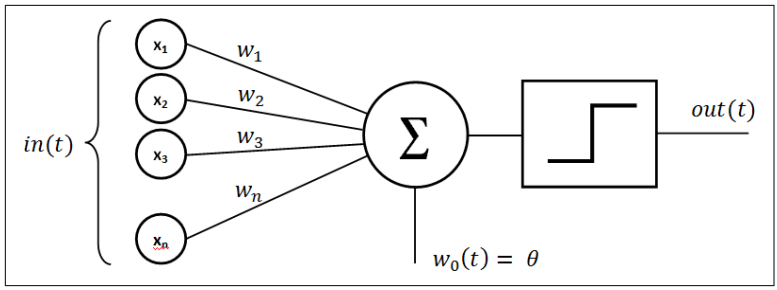


Figure 1-8. The architecture of a Perceptron

Once the Perceptron has completed the inductive learning process, we have a network of voltage levels which implicitly describe the rule system. We call this a *distributed representation*. It can produce the correct outputs, but it is hard to look at a distributed representation and understand the rules explicitly. Like in our own neural networks, the rules are represented implicitly or impressionistically. Nonetheless, they serve the desired purpose.

Though Perceptrons are capable of performing inductive learning on simple systems, they are not capable of solving linearly inseparable problems. To solve this kind of problem, we need to account for interdependent relationships between attributes. In a sense, we can think of an interdependency as being a kind of attribute in itself. Yet, in complex data, it is often very difficult to spot interdependencies simply by looking at the data. Therefore, we need some way of allowing the learning system to discover and account for these interdependencies on its own. This can be done by adding one or more layers of nodes between the inputs and outputs. The express purpose of these "hidden" nodes is to characterize the interdependencies that may be concealed within the relationships between the data's concrete (or "visible") attributes. The addition of these hidden nodes makes the inductive learning process significantly more complex.

The *backpropagation* algorithm, which was developed in the late 1960s but not fully utilized until a 1986 paper by David Rumelhart et al.,[4] can perform inductive learning for linearly inseparable prob-

4 David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, "Learning representations by back-propagating errors," Cognitive Modeling 5, no. 3 (1988): 1.

lems. Readers interested in learning more about these ideas should refer to the section "Going Further" on page 67.

Common Analogies for Machine Learning

Biological systems

When Leonardo da Vinci set out to design a flying machine, he naturally looked for inspiration in the only flying machines of his time: winged animals. He studied the stabilizing feathers of birds, observed how changes in wing shape could be used for steering, and produced numerous sketches for machines powered by human "wing flapping."

Ultimately, it has proven more practical to design flying machines around the mechanism of a spinning turbine than to directly imitate the flapping wing motion of birds. Nevertheless, from da Vinci onward, human designers have pulled many key principles and mechanisms for flight from their observations of biological systems. Nature, after all, had a head start in working on the problem and we would be foolish to ignore its findings.

Similarly, since the only examples of intelligence we have had access to are the living things of this planet, it should come as no surprise that machine learning researchers have looked to biological systems for both the guiding principles and specific design mechanisms of learning and intelligence.

In a famous 1950 paper, "Computing Machinery and Intelligence,"[5] the computer science luminary Alan Turing pondered the question of whether machines could be made to think. Realizing that "thought" was a difficult notion to define, Turing proposed what he believed to be a closely related and unambiguous way of reframing the question: "Are there imaginable digital computers which would do well in the *imitation game*?" In the proposed game, which is now generally referred to as a *Turing Test*, a human interrogator poses written questions to a human and a machine. If the interrogator is unable to determine which party is human based on the responses to these questions, then it may be reasoned that the machine is intelligent. In the framing of this approach, it is clear that a system's simi-

5 Turing, A. M. "Computing Machinery and Intelligence." *Mind* 59.236 (1950): 433-60.

larity to a biologically produced intelligence has been a central metric in evaluating machine intelligence since the inception of the field.

In the early history of the field, numerous attempts were made at developing analog and digital systems that simulated the workings of the human brain. One such analog device was the Homeostat, developed by William Ross Ashby in 1948, which used an electromechanical process to detect and compensate for changes in a physical space in order to create stable environmental conditions. In 1959, Herbert Simon, J.C. Shaw, and Allen Newell developed a digital system called the General Problem Solver, which could automatically produce mathematical proofs to formal logic problems. This system was capable of solving simple test problems such as the Tower of Hanoi puzzle, but did not scale well because its search-based approach required the storage of an intractable number of combinations in solving more complex problems.

As the field has matured, one major category of machine learning algorithms in particular has focused on imitating biological learning systems: the appropriately named *Artificial Neural Networks* (ANNs). These machines, which include Perceptrons as well as the deep learning systems discussed later in this text, are modeled after but implemented differently from biological systems. See Figure 1-9.

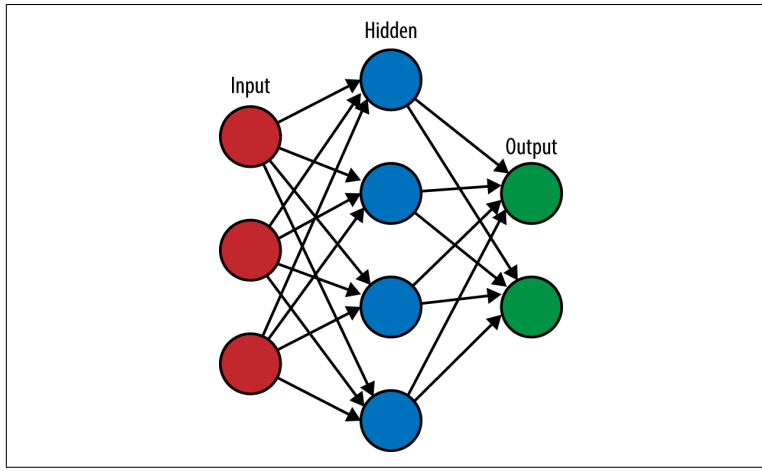


Figure 1-9. The simulated neurons of an ANN

Instead of the electrochemical processes performed by biological neurons, ANNs employ traditional computer circuitry and code to

produce simplified mathematical models of neural architecture and activity. ANNs have a long way to go in approaching the advanced and generalized intelligence of humans. Like the relationship between birds and airplanes, we may continue to find practical reasons for deviating from the specific mechanisms of biological systems. Still, ANNs have borrowed a great many ideas from their biological counterparts and will continue to do so as the fields of neuroscience and machine learning evolve.

Thermodynamic systems

One indirect outcome of machine learning is that the effort to produce practical learning machines has also led to deeper philosophical understandings of what learning and intelligence really are as phenomena in nature. In science fiction, we tend to assume that all advanced intelligences would be something like ourselves, since we have no dramatically different examples of intelligence to draw upon.

For this reason, it might be surprising to learn that one of the primary inspirations for the mathematical models used in machine learning comes from the field of Thermodynamics, a branch of physics concerned with heat and energy transfer. Though we would certainly call the behaviors of thermal systems complex, we have not generally thought of these systems as holding a strong relation to the fundamental principles of intelligence and life.

From our earlier discussion of inductive reasoning, we may see that learning has a great deal to do with the gradual or iterative process of finding a balance between many interrelated factors. The conceptual relationship between this process and the tendency of thermal systems to seek equilibrium has allowed machine learning researchers to adopt some of the ideas and equations established within thermodynamics to their efforts to model the characteristics of learning.

Of course, what we choose to call "intelligence" or "life" is a matter of language more than anything else. Nevertheless, it is interesting to see these phenomena in a broader context and understand that nature has a way of reusing certain principles across many disparate applications.

Electrical systems

By the start of the twentieth century, scientists had begun to understand that the brain's ability to store memories and trigger actions in the body was produced by the transmission of electrical signals between neurons. By mid-century, several preliminary models for simulating the electrical behaviors of an individual neuron had been developed, including the Perceptron. As we saw in the "Biological systems" on page 15 section, these models have some important similarities to the logic gates that comprise the basic building blocks of electronic systems. In its most basic conception, an individual neuron collects electrical signals from the other neurons that lead into it and forwards the electrical signal to its connected output neurons when a sufficient number of its inputs have been electrically activated.

These early discoveries contributed to a dramatic overestimation of the ease with which we would be able to produce a true artificial intelligence. As the fields of neuroscience and machine learning have progressed, we have come to see that understanding the electrical behaviors and underlying mathematical properties of an individual neuron elucidates only a tiny aspect of the overall workings of a brain. In describing the mechanics of a simple learning machine somewhat like a Perceptron, Alan Turing remarked, "The behavior of a machine with so few units is naturally very trivial. However, machines of this character can behave in a very complicated manner when the number of units is large."[6]

Despite some similarities in their basic building blocks, neural networks and conventional electronic systems use very different sets of principles in combining their basic building blocks to produce more complex behaviors. An electronic component helps to route electrical signals through explicit logical decision paths in much the same manner as conventional computer programs. Individual neurons, on the other hand, are used to store small pieces of the distributed representations of inductively approximated rule systems.

So, while there is in one sense a very real connection between neural networks and electrical systems, we should be careful not to think of

6 Alan Mathison Turing, "Intelligent Machinery," in *Mechanical Intelligence*, ed. D. C. Ince (Amsterdam: North-Holland, 1992), 114.

brains or machine learning systems as mere extensions of the kinds of systems studied within the field of electrical engineering.

Ways of Learning

In machine learning, the terms *supervised*, *unsupervised*, *semi-supervised*, and *reinforcement learning* are used to describe some of the key differences in how various models and algorithms learn and what they learn about. There are many additional terms used within the field of machine learning to describe other important distinctions, but these four categories provide a basic vocabulary for discussing the main types of machine learning systems:

Supervised learning procedures are used in problems for which we can provide the system with example inputs as well as their corresponding outputs and wish to induce an implicit approximation of the rules or function that governs these correlations. Procedures of this kind are "supervised" in the sense that we explicitly indicate what correlations should be found and only ask the machine how to substantiate these correlations. Once trained, a supervised learning system should be able to predict the correct output for an input example that is similar in nature to the training examples, but not explicitly contained within it. The kinds of problems that can be addressed by supervised learning procedures are generally divided into two categories: *classification* and *regression* problems. In a classification problem, the outputs relate to a set of discrete categories. For example, we may have an image of a handwritten character and wish to determine which of 26 possible letters it represents. In a regression problem, the outputs relate to a real-valued number. For example, based on a set of financial metrics and past performance data, we may try to guess the future price of a particular stock.

Unsupervised learning procedures do not require a set of known outputs. Instead, the machine is tasked with finding internal patterns within the training examples. Procedures of this kind are "unsupervised" in the sense that we do not explicitly indicate what the system should learn about. Instead, we provide a set of training examples that we believe contains internal patterns and leave it to the system to discover those patterns on its own. In general, unsupervised learning can provide assistance in our efforts to understand extremely complex systems whose internal patterns may be too complex for humans to discover on their own. Unsupervised learning can also be used to produce *generative* models, which can, for

example, learn the stylistic patterns in a particular composer's work and then generate new compositions in that style. Unsupervised learning has been a subject of increasing excitement and plays a key role in the deep learning renaissance, which is described in greater detail below. One of the main causes of this excitement has been the realization that unsupervised learning can be used to dramatically improve the quality of supervised learning processes, as discussed immediately below.

Semi-supervised learning procedures use the automatic feature discovery capabilities of unsupervised learning systems to improve the quality of predictions in a supervised learning problem. Instead of trying to correlate raw input data with the known outputs, the raw inputs are first interpreted by an unsupervised system. The unsupervised system tries to discover internal patterns within the raw input data, removing some of the noise and helping to bring forward the most important or indicative features of the data. These distilled versions of the data are then handed over to a supervised learning model, which correlates the distilled inputs with their corresponding outputs in order to produce a predictive model whose accuracy is generally far greater than that of a purely supervised learning system. This approach can be particularly useful in cases where only a small portion of the available training examples have been associated with a known output. One such example is the task of correlating photographic images with the names of the objects they depict. An immense number of photographic images can be found on the Web, but only a small percentage of them come with reliable linguistic associations. Semi-supervised learning allows the system to discover internal patterns within the full set of images and associate these patterns with the descriptive labels that were provided for a limited number of examples. This approach bears some resemblance to our own learning process in the sense that we have many experiences interacting with a particular kind of object, but a much smaller number of experiences in which another person explicitly tells us the name of that object.

Reinforcement learning procedures use rewards and punishments to shape the behavior of a system with respect to one or several specific goals. Unlike supervised and unsupervised learning systems, reinforcement learning systems are not generally trained on an existent dataset and instead learn primarily from the feedback they gather through performing actions and observing the consequences. In sys-

tems of this kind, the machine is tasked with discovering behaviors that result in the greatest reward, an approach which is particularly applicable to robotics and tasks like learning to play a board game in which it is possible to explicitly define the characteristics of a successful action but not how and when to perform those actions in all possible scenarios.

What Is Deep Learning?

From Alan Turing's writings onwards, the history of machine learning has been marked by alternating periods of optimism and discouragement over the field's prospects for applying its conceptual advancements to practical systems and, in particular, to the construction of a general-purpose artificial intelligence. These periods of discouragement, which are often called *AI winters*, have generally stemmed from the realization that a particular conceptual model could not be easily scaled from simple test problems to more complex learning tasks. This occurred in the 1960s when Marvin Minsky and Seymour Papert conclusively demonstrated that perceptrons could not solve linearly inseparable problems. In the late 1980s, there was some initial excitement over the backpropagation algorithm's ability to overcome this issue. But another AI winter occurred when it became clear that the algorithm's theoretical capabilities were practically constrained by computationally intensive training processes and the limited hardware of the time.

Over the last decade, a series of technical advances in the architecture and training procedures associated with artificial neural networks, along with rapid progress in computing hardware, have contributed to a renewed optimism for the prospects of machine learning. One of the central ideas driving these advances is the realization that complex patterns can be understood as hierarchical phenomena in which simple patterns are used to form the building blocks for the description of more complex ones, which can in turn be used to describe even more complex ones. The systems that have arisen from this research are referred to as "deep" because they generally involve multiple layers of learning systems which are tasked with discovering increasingly abstract or "high-level" patterns. This approach is often referred to as *hierarchical feature learning*.

As we saw in our earlier discussion of the process of recognizing a human face, learning about a complex idea from raw data is chal-

lenging because of the immense variability and noise that may exist within the data samples representing a particular concept or object.

Rather than trying to correlate raw pixel information with the notion of a human face, we can break the problem down into several successive stages of conceptual abstraction (see Figure 1-10). In the first layer, we might try to discover simple patterns in the relationships between individual pixels. These patterns would describe basic geometric components such as lines. In the next layer, these basic patterns could be used to represent the underlying components of more complex geometric features such as surfaces, which could be used by yet another layer to describe the complex set of shapes that compose an object like a human face.

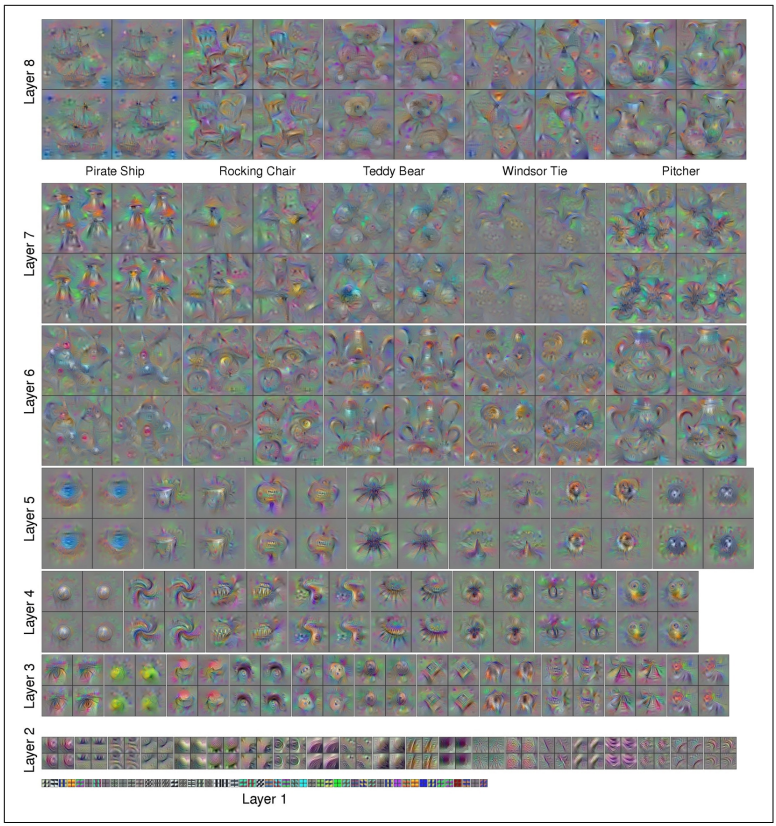


Figure 1-10. Hierarchical feature layers of an image recognition convolutional neural network (image courtesy of Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, "Understanding neural networks through deep visualization," presented at the Deep Learning Workshop, International Conference on Machine Learning (ICML), 2015)

As it turns out, the backpropagation algorithm and other earlier machine learning models are capable of achieving results comparable to those associated with more recent deep learning models, given sufficient training time and hardware resources. It should also be noted that many of the ideas driving the practical advances of deep learning have been mined from various components of earlier models. In many ways, the recent successes of deep learning have less to do with the discovery of radically new techniques than with a series of subtle yet important shifts in our understanding of various component ideas and how to combine them. Nevertheless, a well-timed

shift in perspective, coupled with diminishing technical constraints, can make a world of difference in exposing a series of opportunities that may have been previously conceivable but were not practically achievable.

As a result of these changes, engineers and designers are poised to approach ever more complex problems through machine learning. They will be able to produce more accurate results and iterate upon machine-learning-enhanced systems more quickly. The improved performance of these systems will also enable designers to include machine learning functionality that would have once required the resources of a supercomputer in mobile and embedded devices, opening a wide range of new applications that will greatly impact users.

As these technologies continue to progress over the next few years, we will continue to see radical transformations in an astounding number of theoretical and real-world applications from art and design to medicine, business, and government.

Enhancing Design with Machine Learning

Parsing Complex Information

Computers have long offered peripheral input devices like microphones and cameras, but despite their ability to transmit and store the data produced by these devices, they have not been able to understand it. Machine learning enables the parsing of complex information from a wide assortment of sources that were once completely indecipherable to machines.

The ability to recognize spoken language, facial expressions, and the objects in a photograph enables designers to transcend the expressive limitations of traditional input devices such as the keyboard and mouse, opening an entirely new set of interaction paradigms that will allow users to communicate ideas in ever more natural and intuitive ways.

In the sections that follow, we will look more closely at some of these opportunities. But before doing so, it should be noted that some of the possibilities we will explore currently require special hardware or intensive computing resources that may not be practical or accessible in all design contexts at this time. For example, the Microsoft

Kinect, which allows depth sensing and body tracking, is not easily paired with a web-based experience. The quickly evolving landscape of consumer electronics will progressively deliver these capabilities to an ever wider spectrum of devices and platforms. Nevertheless, designers must take these practical constraints into consideration as they plan the features of their systems.

Enabling Multimodal User Input

In our everyday interactions with other people, we use hand gestures and facial expressions, point to objects and draw simple diagrams. These auxiliary mechanisms allow us to clarify the meaning of ideas that do not easily lend themselves to verbal language. They provide subtle cues, enriching our descriptions and conveying further implications of meaning like sarcasm and tone. As Nicholas Negroponte said in *The Architecture Machine*, "it is gestures, smiles, and frowns that turn a conversation into a dialogue."[7]

In our communications with computers, we have been limited to the mouse, keyboard, and a much smaller set of linguistic expressions. Machine learning enables significantly deeper forms of linguistic communication with computers, but there are still many ideas that would be best expressed through other means—visual, auditory, or otherwise. As machine learning continues to make a wider variety of media understandable to the computer, designers should begin to employ "multimodal" forms of human-computer interaction, allowing users to convey ideas through the optimal means of communication for a given task. As the saying goes, "a picture is worth a thousand words"—at least when the idea is an inherently visual one.

For example, let's say the user needed a particular kind of screwdriver but didn't know the term "Phillips head." Previously, he might have tried Googling a variety of search terms or scouring through multiple Amazon listings. With multimodal input, the user could instead tell the computer, "I'm looking for a screwdriver that can turn this kind of screw" and then upload a photograph or draw a sketch of it. The computer would then be able to infer which tool was needed and point the user to possible places to purchase it.

---

7 Negroponte, Nicholas. *The Architecture Machine*. Cambridge, MA: M.I.T., 1970. 11. Print.

By conducting each exchange in the most appropriate modality, communication is made more efficient and precise. Interactions between user and machine become deeper and more varied, making the experience of working with a computer less monotonous and therefore more enjoyable. Complexity and nuance are preserved where they might otherwise have been lost to a translation between media.

This heightened expressivity, made possible by machine learning's ability to extract meaning from complex and varied sources, will dramatically alter the nature of human&8#211;computer interactions and require designers to rethink some of the longstanding principles of user interface and user experience design.

New Modes of Input

Visual Inputs

The visible world is full of nuanced information that is not easily conveyed through other means. For this reason, extracting information from images has been one of the primary applied goals throughout the history of machine learning. One of the earliest applications in this domain is *optical character recognition*—the task of decoding textual information, either handwritten or printed, from photographic sources (see Figure 1-11). This technology is used in a wide range of real-world applications from the postal service's need to quickly decipher address labels to Google's effort to digitize and make searchable the world's books and newspapers. The pursuit of optical character recognition systems has helped to drive machine learning research in general and has remained as one of the key test problems for assessing the performance of newly invented machine learning algorithms.

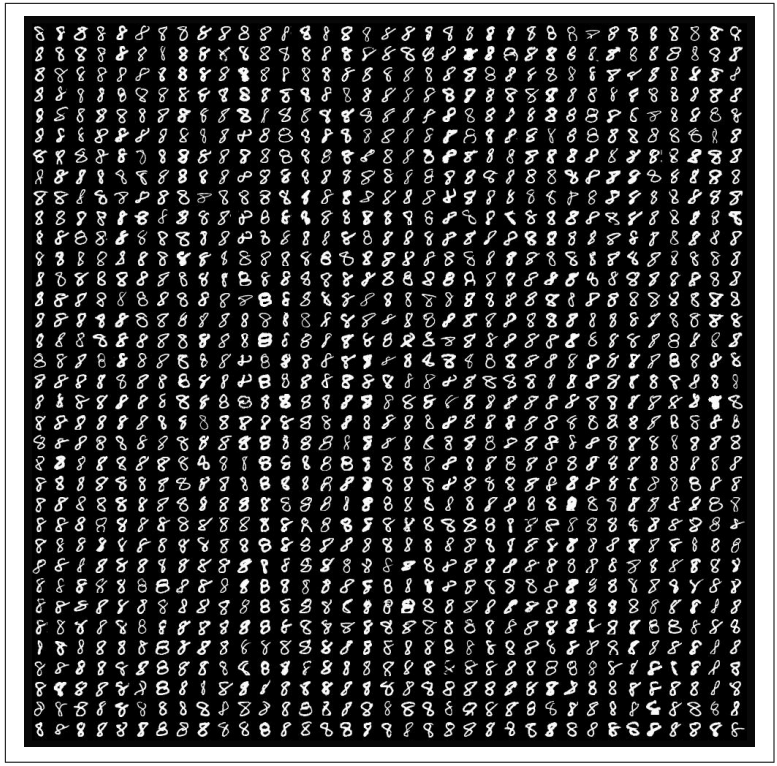


Figure 1-11. Handwritten '8' digits from the MNIST database

More recently, researchers have turned their attention to more complex visual learning tasks, many of which center around the problem of identifying objects in images. The goals of these systems range in both purpose and complexity. On the simpler end of the spectrum, *object recognition* systems are used to determine whether a particular image contains an object of a specific category such as a human face, cat, or tree. These technologies extend to more specific and advanced functionality, such as the identification of a particular human face within an image. This functionality is used in photo-sharing applications as well as security systems used by governments to identify known criminals.

Going further, *image tagging* and *image description* systems are used to generate keywords or a sentence that describes the contents of an image (see Figure 1-12). These technologies can be used to aid image-based search processes as well as assist visually impaired users to extract information from sources that would be otherwise

inaccessible to them. Further still, *image segmentation* systems are used to associate each pixel of a given image with the category of object represented by that region of the image. In an image of suburban home, for instance, all of the pixels associated with the patio floor would be painted one color while the grass, outdoor furniture and trees depicted in the image would each be painted with their own unique colors, creating a kind of pixel-by-pixel annotation of the image's contents.

Figure 1-12. Example outputs from a neural network trained to produce image descriptions (image courtesy of Karpathy, Andrej, and Li Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015)

Other applications of machine learning to visual tasks include depth estimation and three-dimensional object extraction. These technologies are applicable to tasks in robotics and the development of self-driving cars as well as the automated conversion of conventional two-dimensional movies into stereoscopic ones.

The range of visual applications for machine learning is too vast to enumerate fully here. In general, though, a great deal of machine learning research and applied work has and will continue to be directed to the numerous component goals of turning once impenetrable pixel grids into high-level information that can be acted upon by machines and used to aid users in a wide assortment of complex tasks.

Aural Inputs

Like visual information, auditory information is highly complex and used to convey a wide range of content ranging from human speech to music to bird calls, which are not easily transmitted through other media. The ability for machines to understand spoken language has immense implications for the development of more natu-

ral interaction paradigms. However, the diverse vocal characteristics and speech patterns of different speakers makes this task difficult for machines and even, at times, human listeners. Though a highly reliable *speech-to-text* system can greatly benefit human–computer interactions, even a slightly less reliable system can result in great frustration and lost productivity for the user. Like visual learning systems, immense progress in the complex task of speech recognition has been made in recent years, primarily as a result of breakthroughs in deep learning research. For most applications, these technologies have now matured to the point where their utility generally outweighs any residual imprecision in their capabilities.

Aside from speech recognition, the ability to recognize a piece of music aurally has been another popular area of focus for machine learning research. The Shazam app allows users to identify songs by allowing the software to capture a short snippet of the audio. This system, however, can only identify a song from its original recording rather than allowing users to sing or hum a melody they wish to identify. The SoundHound app offers this functionality, though it is generally less reliable than Shazam's functionality. This is understandable because Shazam can utilize subtle patterns in the recorded audio information to produce accurate recognition results, whereas SoundHound's functionality must attempt to account for the potentially highly imprecise or out-of-tune approximation of the recording by the user's own voice. Both systems, however, provide users with capabilities that would hard to supplant through other means - most readers will be able to recall a time in which they hummed a melody to friends, hoping someone might be able to identify the song. The underlying technologies used by these systems can also be directed towards other audio recognition tasks such as the identification of a bird from its call or the identification of a malfunctioning mechanical system from the noise it makes.

Corporeal Inputs

Body language can convey subtle information about a user's emotional state, augment or clarify the tone of a verbal expression, or be used to specify what object is being discussed through the act of pointing. Machine learning systems, in conjunction with a range of new hardware devices, have enabled designers to provide users with mechanisms for communicating with machines through the endlessly expressive capabilities of the human body. See Figure 1-13.

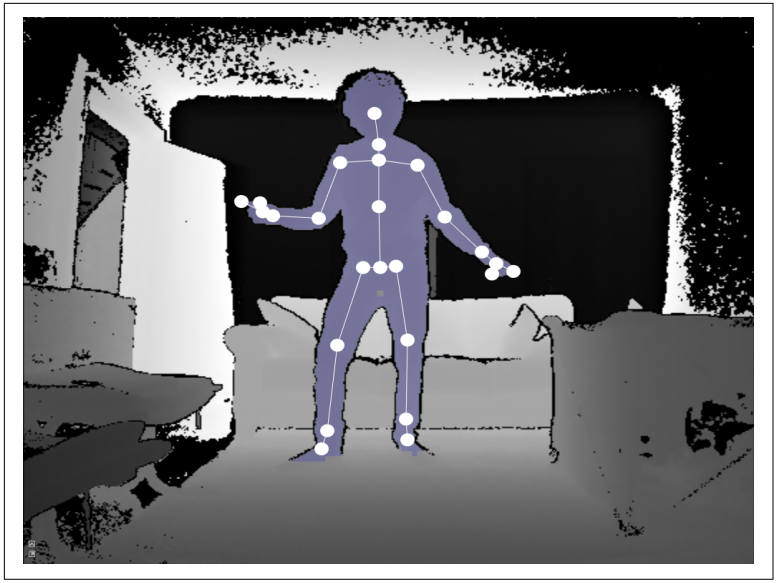


Figure 1-13. Skeleton tracking data produced by the Microsoft Kinect 2 for Windows

Devices such as the Microsoft Kinect and Leap Motion use machine learning to extract information about the location of a user's body from photographic data produced by specialized hardware. The Kinect 2 for Windows allows designers to extract 20 3-dimensional joint positions through its full-body skeleton tracking feature and more than a thousand 3-dimensional points of information through its high-definition face tracking feature. The Leap Motion device provides high-resolution positioning information related to the user's hands.

These forms of input data can be coupled with machine-learning-based gesture or facial expression recognition systems, allowing users to control software interfaces with the more expressive features of their bodies and enabling designers to extract information about the user's mood.

To some extent, similar functionality can be produced using lower-cost and more widely available camera hardware. For the time being, these specialized hardware systems help to make up for the limited precision of their underlying machine learning systems. However, as the capabilities of these machine learning tools quickly advance, the need for specialized hardware in addressing these

forms of corporeal input will be diminished or rendered unnecessary.

In addition to these input devices, health tracking devices like Fitbit and Apple Watch can also provide designers with important information about the user and her physical state. From detecting elevated stress levels to anticipating a possible cardiac event, these forms of user input will prove invaluable in better serving users and even saving lives.

Environmental Inputs

Environmental sensors and Internet-connected objects can provide designers with a great deal of information about users' surroundings, and therefore about the users themselves. The Nest Learning Thermostat (Figure 1-14), for instance, tracks patterns in homeowners' behaviors to determine when they are at home as well as their desired temperature settings at different times of day and during different seasons. These patterns are used to automatically tune the thermostat's settings to meet user needs and make climate control systems more efficient and cost-effective.

Figure 1-14. Nest Learning Thermostat

As Internet-of-Things devices become more prevalent, these input devices will provide designers with new opportunities to assist users in a wide assortment of tasks from knowing when they are out of milk to when their basement has flooded.

Abstract Inputs

In addition to physical forms of input, machine learning allows designers to discover implicit patterns within numerous facets of a user's behavior. These patterns carry inherent meanings, which can be learned from and acted upon, even if the user is not expressly aware of having communicated them. In this sense, these implicit patterns can be thought of as input modalities that, in practice, serve a very similar purpose to the more tangible input modes described above.

Mining behavioral patterns through machine learning can help designers to better understand users and serve their needs. At the

same time, these patterns can also help designers to understand the products or services they offer as well as the implicit relationships between these offerings. Behavioral patterns can be mined in relation to an individual user or aggregated from the collective behaviors of numerous users.

One form of pattern mining that can be useful in serving an individual user as well as improving the overall system is the discovery of frequently coupled behaviors within the sequence of actions performed by the user. For example, a user may purchase milk whenever he buys breakfast cereal. Noticing this pattern gives designers the opportunity to construct interface mechanisms that will allow the user to address his shopping needs more easily and efficiently. When the user adds cereal to the shopping cart, a modal interface suggesting the purchase of milk could be presented to him. Alternately, these two items could be shown in proximity to one another within the interface, despite the fact that these two products would generally be situated within two different areas of the store. Rather than presenting the user with separate interfaces for each item, the system could instead dynamically generate a single interface element that would allow the user to purchase these frequently coupled items with one click. In addition to benefiting the user's shopping experience and enabling multiuser recommendation engines, the system's knowledge of these correlated behaviors can be used to make the system itself more efficient, aiding business processes like inventory estimation.

Mining user behavior patterns can also help businesses to better understand who their customers are. If a user frequently purchases diapers, for instance, it is a near certainty that the user is a parent. This kind of auxiliary knowledge of the user can help designers to produce interfaces that better address their target customer demographics and influence business and marketing decisions such as the determination of which advertising venues will yield the greatest influx of new customers. Designers should be careful, however, to not make assumptions about users that may embarrass or offend them by characterizing them in ways that conflict with the public persona they wish to convey.

In one famous incident, the retailer Target used purchasing patterns to determine whether a given user was pregnant so that the store

could better target this much sought-after category of customer.[8] Though this practice may be welcomed by some, in at least one case it created an uncomfortable situation when an angry father came to a Target store demanding to know why his high-school-aged daughter had received numerous coupons targeted at expectant mothers. The man had not been aware that his daughter was pregnant and upon learning the truth from his daughter, apologized to the retailer. Nevertheless, these kinds of customer insights can be unsettling and require great care in their treatment by designers. Issues of this kind will be discussed in greater detail in the section "Mitigating Faulty Assumptions" on page 52.

When handled with care, however, user insights can provide great value to customers and businesses alike. Understanding a user's behavioral patterns can also aid security processes such as the detection of fraudulent use of a customer's account information. Credit card companies and some retailers regularly use consumer purchasing patterns to assess whether a given purchase is fraudulent by checking whether the geographic location of the transaction and the items purchased are in line with the customer's history. If an anomaly is detected, the transaction will be blocked and the customer will be notified that their financial information may be compromised.

Creating Dialogue

Conventional user interfaces tend to rely upon menu systems as the primary means of organizing the set of features available to the user. One positive aspect of this approach is that it provides a clear and explicit mechanism for the user to explore the system and learn what is possible within it. Hierarchical menus enable users to quickly focus their search for a specific feature by curating the system's functionality into neatly delineated, domain-specific groupings. If the user is unaware of a particular feature, its proximity to other more familiar functions may lead the user to experiment with it, expanding her knowledge of the system in an organic manner.

The downside to hierarchical menu systems, however, is that once users have learned the system's capabilities, they still have to spend a great deal of time navigating menus in order to reach commonly used features. This tedious labor is sometimes mediated by the soft-

8 *http://www.nytimes.com/2012/02/19/magazine/shopping-habits.html*

ware's offering of keyboard shortcuts. But there is a limit to the number of shortcuts that can be plausibly utilized, and this approach is not easily embedded in web and mobile interfaces.

With each additional feature, conventional menu systems become increasingly complex and hard to navigate. For professional tools like 3D-modeling and video-editing software, it may be reasonable to expect the user to commit to a steep learning curve. But for intelligent assistants and other applications with broad feature sets aimed at the average user, it would be impractical or counterproductive to present the system's functionality through a menu system. Imagine what Siri would look like if it were to take this approach!

Programmers may be familiar with a different mechanism for facilitating interaction: the "read-evaluate-print loop" (or REPL). In this kind of interface, the programmer issues a specific textual command, the machine performs it, prints the output, and waits for the user to issue another command. This back-and-forth dialogue takes the familiar form of a messaging app and allows programmers to work more quickly and dynamically by circumventing the need for the tedious and incessant navigation of menu hierarchies.

This conversational format also provides an ideal medium for facilitating the kinds of multimodal interactions described in the previous section. In this machine-learning-enhanced context, we can imagine many vibrant and varied exchanges between the user and machine. Much like a whiteboarding session held by human collaborators, the user and software would take turns addressing aspects of the task at hand, solving its component problems and clarifying their ideas through an assortment of verbal expressions, diagrams, and gestures.

Assisting Feature Discovery

Despite its many advantages, an open-ended dialogue does little to make the system's capabilities known to the user. To make the most of a REPL programming interface, the user must hold prior knowledge of the available features or make frequent reference to the documentation, which is likely to negate any efficiency gains over menu-based workflows. For intelligent interfaces, offloading the process of feature discovery to auxiliary documentation would be even less practical.

The potential limitations of a conversational user interface can be seen in our interactions with first-generation intelligent assistants such as Siri, Cortana, and Echo(Figure 1-15). In this context, feature discovery often consists of the user scouring his or her mind for phrases that might lead to some amusing or useful response from the software.

Figure 1-15. Amazon's Echo

One solution to the problem of feature discovery that is offered by many intelligent assistants is to provide the user with a set of example inputs that demonstrate a variety of features available within the system. The computational knowledge engine WolframAlpha embeds examples directly into its interface, helping new users to become acquainted with the system's mode of interaction and the kinds of knowledge it is able to access. These well-curated examples continue to engage even seasoned users, inviting further exploration into the entertaining or informative pathways they reveal.

Demonstrating an application's full range of features through a series of embedded examples can, however, begin to make the software look suspiciously like a reference manual. Rather than handpicking a static group of examples, machine learning or conventional programmatic logic can be used to dynamically select examples that are contextually relevant to the user's activity. Going further, machine intelligence could be used to generate speculative extensions of the user's trajectory, providing a series of possible next steps in a range of conceptual directions for the user to consider. This mechanism would not only extend the user's understanding of the interface and its possibilities, but also enrich his or her fluency with the task or domain of interest at hand.

Another common mechanism for helping users to locate features within a conversational interface is to offer suggested meanings when their statements are not clear to the system. This can be helpful in correcting the improperly formed expression of a command that is already more-or-less known to the user. But if the user is completely unaware of some potentially advantageous feature, he will not think to ask for it in the first place, and this mechanism can provide no assistance.

We could, in theory, defer the problem of feature discovery to the march of progress in technology and assume that computers will eventually be intelligent enough to perform any task that could conceivably be asked of them. Yet, the machine's ability to handle any request would not inherently translate to the user's awareness of every possible request that might prove useful to his goals. Furthermore, the user may not be familiar enough with the problem domain to develop a set of goals in the first place.

Think for a moment about what question you might ask a Nobel laureate in chemistry. Without a strong working knowledge of the

field, you would be unlikely to formulate a question that took full advantage of her expertise. To make the most of this opportunity, you would like need some guidance in finding an entry point to a fruitful line of inquiry.

To assist users in performing complex or domain-specific tasks, designers should strive to build systems that not only respond to user requests but also enrich the user's understanding of the domain itself. It is not enough for intelligent interfaces to offer an infinite variety of features; they must also find ways of connecting the user with the possibilities they contain.

Getting Acquainted

Perhaps the greatest challenge of any conversation is knowing where to begin. When you meet someone new, you cannot be sure that your personal experiences, professional background, or even the idioms and gestures you use will overlap with theirs. For this reason, we often begin with simple pleasantries and then attempt to establish common ground by posing basic questions about the other person's interests, line of work, and family before diving into more specific subjects and inquiries.

Similarly, in designing conversational user interfaces, we must establish common ground with users and introduce them to the system's capabilities and modes of interaction, since they are not explicitly detailed by a menu system.

A blank page can be stifling and may lead some users to lose interest immediately. For this reason, it is important to have the user get his feet wet as quickly as possible. Rather than simply showcasing example inputs that demonstrate a few basic features, the application could instead ask the user to perform a simple task that employs one of these features. This gives the user an initial confidence boost and helps to get the conversational juices flowing.

As the user gains traction through these initial exercises, the software can gradually become less proactive in soliciting specific actions and allow the user to explore the tool on his or her own terms. Over time, the system can continue to suggest new features to the user. But designers should be careful not to bombard users with too many new ideas as they become acquainted with the system.

Once acclimated, the user's own explorations and activities can help the system to understand what features would be most useful to her. Subsequently, if the user is idle or appears to be at an impasse, the system can suggest contextually relevant new activities or features that have never been explored by the user. This form of assistance will be discussed further in the section "Designing Building Blocks" on page 43.

The user should, of course, always have the ability to opt out of these suggestions or redirect them toward more relevant functionality. Before leaving these introductory exercises, it is also important to make sure that users know how to get help from the system when they need it. Many new users will be eager to dive into the software as quickly as possible, leading them to rush through these important on-boarding exercises. In a conversational interface, an obvious and natural mechanism for seeking assistance from the system would be to use a keyword like "Help!" In some cases, however, the user's confusion may come from a lack of familiarity or comfort with the open-ended nature of a conversational interface. For this reason, it may be valuable to provide a concrete and persistent interface element such as a help button that allows the user to quickly and unambiguously reach out for assistance.

Seeking Clarity

In an open-ended dialogue with an intelligent interface, it is only natural and perhaps even desirable for the user to develop the impression that anything is possible within the system. In most instances, however, this will not be the case. Though intelligent assistants give the appearance of broad conceptual versatility, their true capabilities are generally still limited to one or a handful of specific domains of functionality. Their ability to comprehend human language and map it to a particular function is also often somewhat brittle.

Designers can help to mediate these technical limitations by providing conversational cues that lead the user away from broad or ambiguous statements, which cannot be easily parsed by the machine. A key component of this is to design for interactions that establish realistic expectations from the user and clearly indicate the kind of information that is being requested either through example or brief description.

If an application prompted you to "say what's on your mind," it would be disappointing or embarrassing to dive into a longwinded and enthusiastic description of your ideas only to have the machine respond with a terse "I'm sorry, I don't understand."

Helping the user to focus on a specific task or to communicate a singular point of information is often most difficult at the beginning of a new interaction. The user may have several impressionistic or partially formed goals, but may not have sufficient clarity to articulate a clear point of entry to the system. The absence of accumulated contextual information that could otherwise be derived from the user's most recent actions also makes it more difficult for the machine to infer the user's intent at the outset of an interaction.

Rather than starting with an open-ended question, the system could instead pose a specific yet basic one that will help to establish context and get the conversation going so that further details can be solicited in subsequent exchanges. In many cases, it may be helpful to provide users with a few categorical options that will localize their goals to a specific subset of the system's overall functionality. In the context of an e-commerce site, the system might open the conversation by asking whether the user is looking to purchase a specific item, browse for items in a particular section of the store, or make a return. Once the broad context has been established, the system can seek further details from the user by posing questions that extend organically from this point of entry. If the user indicated an interest in buying a specific item, the system might pose a series of questions about the desired size, materials, or optional accessories associated with that item.

There is nothing more discouraging than having your ideas fall flat with no indication of what was confusing or how to better communicate your intended meaning. Perhaps the user's expression was completely outside the machine's domain of understanding. Or perhaps a single word choice or change of phrasing would have made the statement clear. But without any feedback from the machine, the user will be left in frustration to guess at the problem.

When the user's input has been understood, the system should restate what it has understood before moving on to subsequent exchanges. If the system has not understood the user, it should try to indicate the nature of the problem to the best of its ability.

Many machine learning and natural language processing platforms can be configured to return multiple speculative interpretations of the user's expression alongside *confidence scores* that numerically indicate the machine's certainty about the correctness of each interpretation it offers. Whenever possible, intelligent applications should try to correlate these speculative interpretations with broader contextual knowledge of the user's activities in order to eliminate the least plausible options. If this approach does not uncover a single unambiguous meaning, designers should consider presenting a handful of the most highly ranked interpretations to the user directly. Exposing users to these speculative interpretations will help them to understand the nature of the problem and rephrase or redirect a statement.

Breaking Interactions into Granular Exchanges

One of the most important things designers can do to facilitate successful exchanges between users and intelligent interfaces is to the help users break multistage workflows and their component interactions into small conceptual parcels. Simple expressions that communicate an individual command or point of information are more easily understood by machine learning and natural language processing systems than complex, multifaceted statements. Designers can help the user to deliver concise statements by creating interfaces and workflows that lead the user through a series of simple exercises or decision points that each address a single facet of a much larger and more complex task.

An excellent example of this approach is 20Q, an electronic version of the game Twenty Questions. Like the original road trip game, 20Q asks the user to think of an object or famous person and then poses a series of multiple choice questions in order to discover what the user has in mind (see Figure 1-16).

The first question posed in this process is always:

"Is it classified as Animal, Vegetable, Mineral, or Concept?"

Subsequent questions try to uncover further distinctions that extend from the information that has already been provided by the user. For example, if the answer to the first question were "Animal," then the next question posed might be "Is it a mammal?" If instead the first answer were "Vegetable," the next question might be "Is it usually green?" Each of these subsequent questions can be answered

with one of the following options: Yes, No, Unknown, Irrelevant, Sometimes, Maybe, Probably, Doubtful, Usually, Depends, Rarely, or Partly.

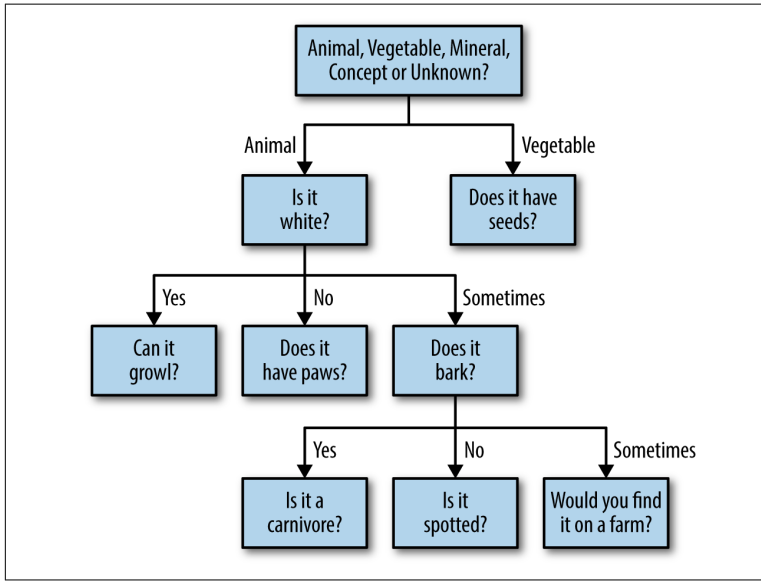


Figure 1-16. A learning decision tree for the game 20Q"

20Q guesses the correct person, place, or thing 80% of the time after 20 questions and 98% of the time after 25 questions. This system uses a kind of machine learning algorithm called a *learning decision tree* (or *classification tree*) to determine the sequence of questions that will lead to the correct answer in the smallest number of steps possible. Using the data generated by previous users' interactions with the system, the algorithm learns the relative value of each question in removing as many incorrect options as possible so that it can present the most important questions to the user first.

For example, if it were already known that the user had a famous person in mind, it would likely be more valuable for the next question to be whether the person is living than whether the person has written a book, because only a small portion of all historic figures are alive today but many famous people have authored a book of one kind or another.

Though none of these questions individually encapsulates the entirety of what the user has in mind, a relatively small number of

well-chosen questions can uncover the correct answer with surprising speed. In addition to aiding the system's comprehension of users' expressions, this process can benefit the users directly in their ability to communicate ideas more clearly and purposefully. This process can also make the task of communicating an idea more playful and engaging for the user.

Aside from guessing games, designers may find this approach to be useful in a wide range of human-computer interactions. At its core, this process can be seen as a mechanism for discovering an optimal path through a large number of interrelated decisions. From this perspective, we can imagine applying this approach to processes like the creation of a financial portfolio or the discovery of new music.

In developing a financial portfolio, users would be presented with a series of questions that relate to their financial priorities and constraints in order to find a set of investments that are well-suited to their long-term goals, risk tolerance, and spending habits. In the context of music or movie discovery, this process may help to overcome some of the limiting factors associated with standard recommendation engines. For example, a user may give a particular film a high rating because it stars one of his favorite actors, even though the film is of a genre that is generally of no interest to the user. From this rating, a recommendation engine is likely to suggest similar films without giving the user the opportunity to point out that the actor rather than the genre was the key factor in his rating. Using an interactive decision tree process, this important distinction could be more easily uncovered by the system.

Designing Building Blocks

The kinds of machine-learning-enhanced workflows described above can provide designers with powerful new tools for making complex tasks more comprehensible, efficient, and enjoyable for the user. At the same time, enabling these modes of interaction may require designers to diverge from at least some of the longstanding conventions of user interface and user experience design.

In the section "Mechanical Induction" on page 11, we discussed inductive learning as a search process. Similarly, any task for which the user has some desired end-state in mind and must discover a series of operations that will reach this goal can be seen as a kind of search. From this perspective, let us imagine a very large map in

which each possible end-state as well as our starting point is represented by a unique set of coordinates.

In this map, each feature of the software can be seen as a road that takes us a certain distance in a particular direction. To navigate this space, we must find a sequence of actions or driving directions that lead from our starting position to the desired destination (see Figure 1-17). A low-level feature would be equivalent to a local road in the sense that it moves us only a short distance within the map, whereas a high-level feature would be more like a highway.

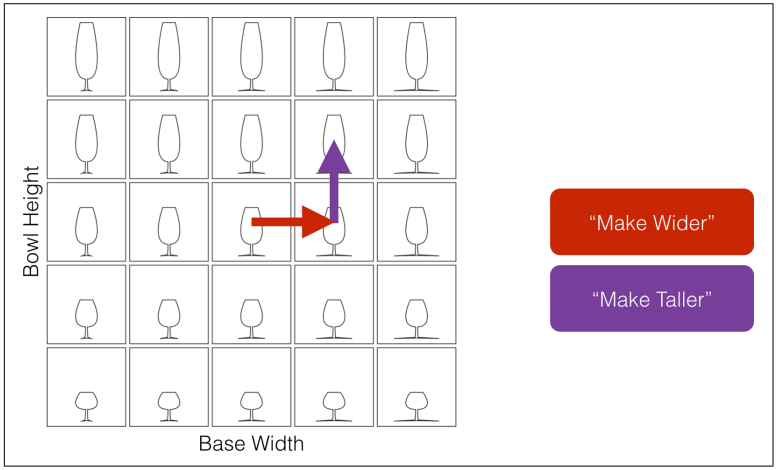


Figure 1-17. A sequence of low-level features related to the design of a wine glass

The nice thing about highways is that they take us great distances with a relatively small number of component actions (see Figure 1-18). The problem, however, is that highways only have exit ramps at commonly visited destinations. To reach more obscure destinations, the driver must take local roads, which requires a greater number of component actions. Ideally, a new highway would be constructed for us whenever we leave home so that we could arrive at any possible destination with a small number of actions (see Figure 1-19). But this is not possible for pre-built high-level interfaces.

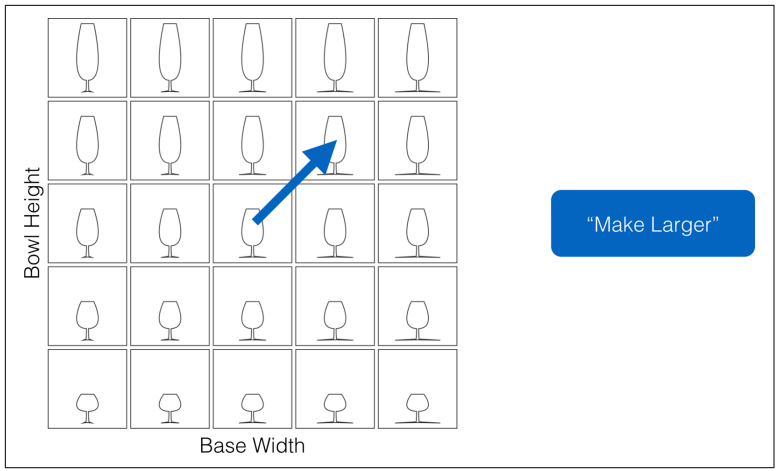


Figure 1-18. A higher-level feature

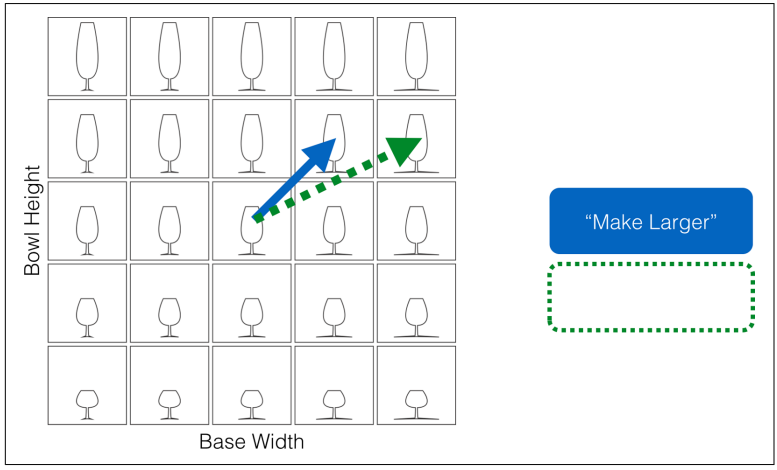


Figure 1-19. A potentially desirable, but non-existent feature

Machine learning allows us to extrapolate a great deal of information about users and what they wish to achieve through the observation of their behaviors. Rather than trying to anticipate the user's needs through a set of prebuilt high-level interfaces, we can instead design systems that learn from the user's engagement with the software. For example, we can discover commonly used sequences of low-level features and then dynamically combine these low-level features into custom, purpose-built features related to the user's current activity within the system.

The behavioral patterns used in the automated production of custom high-level features can be mined from individual users or across many users. Somewhat like recommendation systems that suggest music or movies based on the similarities of users' tastes, the discovery of patterns across numerous users can be employed to suggest relevant features to an individual based on the workflows he or she tends to utilize. This will allow designers to better address the diversity of users and their varied ways of digesting information, making decisions, and interacting with software. It allows designers to meet users where they are rather than asking them to adapt to a singular preordained mentality or workflow offered by a more conventional, static user interface. Additionally, by extrapolating behavioral patterns across many users, designers can better understand the implicit relationships between the features offered within their systems, providing important insights as to how the software can be further developed to better serve the needs of their users.

The high-level features generated through this process can be presented to users in a variety of ways. Based on the recent trajectory of a user's actions, a modal interface offering one or several possible next steps could be presented to the user through a mechanism somewhat like the autocomplete feature offered by some text editing applications. Alternately, a custom-generated feature could be associated with a conventional interface element such as a button. This approach, however, raises questions as to how the user would be made aware of the effect that this newly generated feature would have. In conventional interfaces, buttons are generally associated with a text label or icon that indicates the feature's purpose. For custom-generated features that bundle the functionality of several more granular features, producing labels or icons of this kind may be challenging. Furthermore, it would likely be difficult for the user to keep track of the large and ever-changing iconography of the interface.

To circumvent this challenge, rather than indicating the effect of a given feature through a label or icon, when the user hovers over the interface element associated with the feature, the system could demonstrate its effect directly. For example, a user purchasing a car online might opt to change the paint color. In doing so, a button could be generated so that when the user hovered over it, the view of the car would show a preview of a set of suggested additional changes, such as to the upholstery fabric, trim, and so forth. If the

user clicked the button, these changes would be made, but if the user chose not to click the button and moved the cursor away, the preview of the car would remove these changes.

By adopting this methodology, the designer's role would shift away from the overall curation of high-level functionality and toward the creation of more granular interface elements. The overall curation would emerge from the aggregation of individual points of information communicated by the user. This movement from preset rule systems and interfaces to implicit, intelligently generated ones means that designers would be relinquishing some control of certain aspects of the design. Doing so, however, would enable users to address tasks that have not been explicitly anticipated by the software's designer. This paradigmatic shift would greatly further the most important goal of design: to serve the needs of users.

Acquiring Training Data

A machine learning system is only as good as the data it is trained upon. No matter how powerful an algorithm may be, it cannot extract meaningful patterns if those patterns are not represented by the data. In fact, in some cases, a more powerful machine learning algorithm may produce especially erroneous results from poor training data because, in absence of meaningful patterns, the system may direct its power towards the learning of irrelevant patterns that exist within the dataset's noise.

In general, the quality of a given dataset relates to the following characteristics:

Completeness: The extent to which the data is indicative of the full range of behaviors that is possible within the represented system. For instance, temperature data for a given city would be highly incomplete if it were only recorded on rainy days.

Accuracy: The extent to which the data is true to the real-world behaviors it represents. In other words, the recorded temperature on a particular day should exactly match what the actual temperature was.

Consistency: The extent to which various data points within the set do not conflict with one another. This means that there should not be more than one recorded temperature in the data set for the same time period and location.

Timeliness: The extent to which the data is relevant to the current state of the system. For example, temperature data for a given city from 1900 may not be indicative of more recent patterns in the city's weather.

To achieve these characteristics within a dataset, a great deal of human labor is often required. Accuracy, consistency, and timeliness require careful data collection, curation and maintenance. Completeness is often achieved in part through the sheer volume of examples within the dataset. With a greater number of examples, it is more likely the dataset will account for the full range of possible behaviors. For this reason, it is not uncommon for the datasets used by large-scale machine learning systems to contain hundreds of thousands or millions of training samples. The ImageNet dataset (see Figure 1-20), used to train Google's image classification system, contains over 14 million images, each of which is accompanied by a textual label that identifies which one of over 22,000 categories of objects the image depicts.[9]

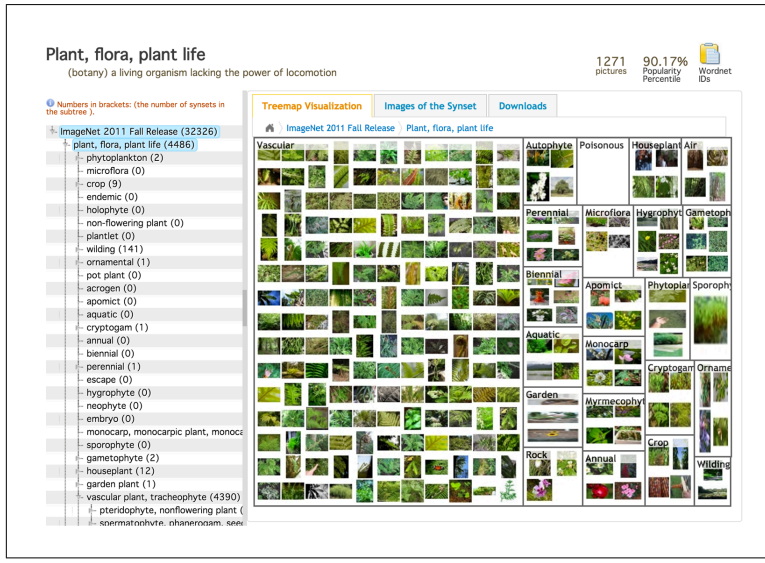


Figure 1-20. A selection of images from the ImageNet dataset

For certain machine learning problems that have received significant attention from the industry and research community, large and

9 *http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf*

well-curated datasets are freely available on the Web. Resources for finding these datasets can be found in the section "Going Further" on page 67. But for machine learning problems that have not been as widely studied, the availability of usable data has been one of the biggest bottlenecks in the advancement of applied machine learning systems.

The Intelligence Feedback Loop

In the previous section, we discussed how interface design can aid users in clearly communicating ideas and information to machine-learning-enhanced systems. Conversely, these same mechanisms can be employed to improve machine learning systems' ability to learn from user-generated data. Through this symbiotic relationship, the aptitudes of machine learning systems and human users can build upon one another, elevating the capabilities of both parties to new heights. One significant example of this symbiosis at work can be seen in the reCAPTCHA system (Figure 1-21), with which many readers will already be familiar.

Figure 1-21. An example reCAPTCHA

CAPTCHAs were developed as a mechanism for verifying that users trying to access a website are human, doing so by asking them to perform tasks that cannot be easily deciphered by machines. Realizing that the performance of these menial tasks would collectively represent a great deal of wasted human effort, Luis von Ahn and several other researchers at Carnegie Mellon University designed a system called reCAPTCHA that would put this effort to good use.

In its early versions, the reCAPTCHA system would present the user with the image of a word that had failed to be deciphered by an optical character recognition system tasked with digitizing printed texts. This image was often paired with the image of a "control" word that had already been deciphered. To gain access to the website, the user would be asked to type the two words. If they correctly entered the control word, it was assumed that the user's entry was valid and they would be given access to the site. Their entry for the previously undeciphered word would then be compared against entries for the

same image performed by several other users. If multiple users' entries for a specific image were in agreement, the entry was assumed to be correct and would be inserted into the digitized version of the originating text.

Google later acquired this technology, which played an important role in the digitization of the *New York Times* archives and the contents of several university libraries under the Google Books project. More recently, reCAPTCHAs have been used to improve the performance of Google's image recognition system by presenting users with a grid of thumbnail images and asking them to select only the images that depict a particular category of object.

This clever idea provides greater security to websites while simultaneously aggregating small bits of human effort to improve the capabilities of machine learning systems that will in turn benefit human users. reCAPTCHA is a relatively straightforward idea, but a powerful and efficient one. The underlying spirit of this idea can be adapted by designers to a wide variety of applications in the development of machine-learning-enhanced systems. Designers can create symbiotic relationships between users and computational systems that benefit both parties while streamlining the arduous task of data collection and cleaning for system developers.

Dealing with Challenges

Designing for Uncertainty

An unreliable system is often worse than no system at all. In our daily lives, we rely upon countless systems that aid us in tasks related to everything from keeping track of our schedules to keeping our families safe. Whatever services these systems may offer, they are only valuable to us if they work consistently. The detrimental effects of a single failure may far outweigh the benefit produced by a thousand instances of the system working correctly.

Designing a flawless system is a tall order in any context. But it is particularly challenging in the context of machine learning, where a system's behaviors are defined probabilistically through the machine's experiential training on a finite number of discrete examples. The machine has no way of knowing whether these training examples are indicative of every possible circumstance it may encounter in real-world usage. Furthermore, if the machine does

encounter unfamiliar conditions, rather than proclaiming its inability to deal with the situation, it may attempt to fit the new data into its existing model and return its results to the user without any indication that something is amiss.

If a machine learning system has been designed to return confidence scores alongside of its predictions, it is likely that the system's attempt to fit incongruous data into an existing model will yield a low confidence score, giving designers the opportunity to catch erroneous results before they reach the user. Unfortunately, though, there can be no explicit guarantee that a perfect storm of conditions, which obfuscates the incongruousness of a particular data sample, will not arise.

Though conventional computer programs can contain bugs, their explicitly encoded logic provides relatively strong assurances that their behaviors will be predictable and repeatable. As computers have become indispensable parts of our daily lives, we have become increasingly accustomed to their predictability and reliability. As we move towards the adoption of intelligent systems, it will be difficult for users to shift their expectations of reliability, even if their reduced reliability comes with the promise of far more advanced functionality.

To mitigate the risks associated with the erroneous behaviors of machine learning systems and provide users with the best possible experiences, designers should take the following steps before releasing their software to the public:

1. Design for interactions in which the system explicitly restates its understanding of the tasks it has been asked to perform, giving users the chance to catch errors and redirect the system's behavior.
2. When possible, provide fallback mechanisms through conventional user interfaces that allow users to circumvent machine-learning-enhanced functionality and perform tasks using explicit logic and interfaces.
3. Perform rigorous testing of the software in as many environments and usage scenarios as possible to uncover possible faults or inconsistencies that may arise from conditions that differ from those of the original development environment. Limited release to an audience of more knowledgeable and error-

tolerant testers may help to uncover circumstantial inconsistencies before wider release to the public.

4. Use all available metrics, namely confidence scores, to assess the validity of included features. Set realistic expectations in how you present a feature and its effectiveness to the user.
5. Resist the temptation to include an impressive-sounding feature if its behavior is too unreliable. This assessment should weigh the complexity and potential value of the feature against the likelihood of its failure. If a feature offers some potentially revolutionary new capability, users may be more willing to accept that it only works 90% of the time.
6. Make users aware of any risks that might accompany their use of the software or a particular feature and allow them to decide for themselves whether these risks are outweighed by the potential benefits of the system's functionality.
7. In cases where a system failure may have extraordinarily serious consequences such as irrevocable damage to the user's property, injury, or death, the value of a feature's inclusion should be weighed with extreme caution and a lawyer should be consulted to assess the risks and liabilities as well as to formulate any necessary disclaimers that will be presented to users.

Mitigating Faulty Assumptions

We all feel uncomfortable having anyone, human or otherwise, make a decisive characterization of our identity that conflicts with the way we see ourselves or wish to be perceived by others. As machine learning continues to reach further into the domain of personality insights and customer preferences, the ability to connect users with products they are likely to find interesting will lead to better customer experiences and more efficient markets, but also open the door to many possible awkward situations.

It is one thing to say to the user, "I think you'll enjoy this upcoming Barry Manilow concert." It is quite another thing to say, "Your tastes are aligned with our 'old fart' user category, therefore I think you'll enjoy this Barry Manilow concert." Rather than explicitly characterizing the user, we can leave the machine learning system's statistical correlations in the statistical realm and behind the scenes. We can make suggestions about things that the user might find interesting

without stating what lead the machine to believe the user would be interested.

Nevertheless, the statistical bases upon which such recommendations are made may inadvertently reflect cultural biases or other faulty assumptions about the user. In the last few years, machine learning systems have made incredible progress in their ability to perform complex tasks like image tagging. The creators of these systems have been excited to share their work with the world as quickly as possible, even when the software was still in an early stage of development. Yet, it is difficult to guard against or even foresee all possible failings of these nascent technologies and, as a result, several embarrassing or offensive incidents have occurred. In the most serious of these incidents, Google's Photos app tagged several black users as "Gorillas."

The upset caused by this incident is more than understandable. However, it is important to remember that the machine learning system involved has no access to or awareness of the complex societal concepts that make this error offensive. The error is owed in part to an imbalance in the set of images used to train the system, a circumstance which Google and other technology companies should seek to rectify. But, as the machine learning expert Andrew Ng said, "… it's obvious this is an innocent rather than deliberate mistake, and just one of millions of mistakes that learning algorithms undoubtedly make every day."[10]

Regardless of the machine's innocence, this error revealed a flaw which created hard feelings, and we should do everything we can to prevent similar incidents from occurring in the future. One component of this is to strive for ever more accurate machine learning models. Another is to work at discovering any implicit biases that may be in the datasets we provide the machines ahead of time. Yet, given their experiential isolation from the human world, it is unlikely that machines will be able to circumvent all possible cultural prejudices in the foreseeable future. Therefore, we must find ways of preventing such errors through design. For instance, designers should consider offering users explicit mechanisms for flagging offensive content, providing guidance as to how they wish to be

10 *https://plus.google.com/113710395888978478005/posts/dZ7pd4zdaiJ*

characterized, or opting out of particular features that are likely to produce faulty assumptions.

Creating Sanity Checks

Since machine learning models have no inherent means of detecting gaps or biases in their own knowledge, designers should look to outside mechanisms in safeguarding against potentially offensive or erroneous behaviors. This may prove difficult in many instances, because one of the primary motivations for using machine learning in the first place is the capacity of these systems to work with information that is too complex or subtle for the explicit logic of conventional computer programs. However, conventional code can provide at least some assistance in spotting obvious issues like the evocation of overtly derogatory language.

Around the time of this writing, Microsoft released a chatbot called Tay, which was designed to respond to users on Twitter and other messaging services. Within a day, the bot was taken offline after posting an astounding number of inflammatory messages. Perhaps the most surprising thing about this incident was that the messages in question did not contain subtly offensive innuendo or easily misconstrued associations; they instead contained overtly derogatory language and sentiments, using widely known racist and sexist terminology as well as references to Adolph Hitler and the like.

As it turns out, the system was trained on user-generated content and was targeted by people who hoped to produce this outcome by inducing the system to learn from their own hateful statements. Yet, this outcome could have been at least partially avoided if the system's designers had used code to check the bot's messages against a dictionary of offensive words before posting to Twitter. This approach can turn into something of a cat-and-mouse game, because users wishing to subvert such safeguards often resort to inventive spellings of words or tactics like replacing the letter "e:" with the numeral "3" so that their use of flagged terminology will go undetected. It is worth noting, however, that many of the offensive terms used in Tay's messages were spelled correctly and could have been easily caught by even most basic of safeguarding mechanisms.

Though somewhat counterintuitive, another approach that might have been taken in this scenario would have been to use a secondary machine learning system that had been specifically trained for the

detection of offensive content. Systems of this kind are used for spam filtering in services like email and user forums. They are also used by marketers in the sentiment analysis tools they employ to better understand comments about their products posted on social networks.

Though neither of these approaches can completely solve the problem, they can at least guard against the most blatant abuses. Aside from these mechanisms, designers should rely upon the collective intelligence of their user base by providing explicit interface elements that allow users to flag offensive or erroneous content for review by system administrators and subsequent integration into the machine learning model. This approach, more than any automated one, will help designers to stay ahead of malicious users and flawed datasets.

Working with Machine Learning Platforms

In theory, machine learning can be applied to any kind of information that contains patterns, so long as those patterns can be sufficiently exemplified by a set of training data. In practice, however, some forms of information can be more readily accessed and applied to real-world design problems than others. For more common machine learning tasks like image tagging and speech-to-text functionality, designers may utilize turn key solutions offered by a variety of *Machine-Learning-as-a-Service* (MLaaS) platforms, which enable straightforward integration with user-facing systems through RESTful APIs and design patterns. In many cases, these MLaaS platforms will also enable the relatively straightforward deployment of machine learning systems trained on custom datasets provided by the designer. For more exotic or domain-specific use cases, designers may look to more customizable open source machine learning toolkits or even fully customized software, which tend to require a deeper technical understanding of the underlying algorithms as well as of the technical issues related to the deployment of such technologies within large-scale user-facing systems.

Machine-Learning-as-a-Service Platforms

Several large technology companies and startups offer high-level machine learning platforms, which provide designers with straightforward access to turnkey solutions or customized training on

designer-provided data. The list of MLaaS platforms is growing quickly. Some of the most popular platforms include: IBM Watson, Amazon Machine Learning, Google Prediction API, Microsoft Azure, BigML, and ClarifAI.

Despite the many advantages of these systems, there are several important downsides that may factor into a designer's decision of whether to use a platform of this kind in building their system. First, the use of these systems comes with recurring costs that will grow with a wider user base. Though the cost of an individual query is generally quite low and bulk rates are available, for a sufficiently large user base, these costs can become prohibitive without a viable revenue model to support the product. Additionally, these platforms generally do not provide a straightforward path for moving a system developed within one MLaaS platform to a competing platform. This platform lock-in may contribute to the long-term cost of ownership and may constrain future innovation within a designer's system. Finally, systems built on top of MLaaS platforms tend to require the user's device to have internet connectivity in order to query the remotely hosted model. This may be limiting in some applications and may incur data usage costs for the user. However, the models associated with many complex machine learning may be too large or computationally intensive to run on user devices, making cloud-based deployment the only viable route regardless of whether a MLaaS or custom machine learning solution has been used.

Turnkey Solutions

If the machine-learning-based functionality you wish to offer within your design is specifically supported by a turnkey feature of one of these platforms, this approach will offer the quickest and easiest path to a deployable user-facing product. Though the specific features offered will differ by platform, many MLaaS platforms offer turnkey solutions for tasks including natural language parsing, language translation, speech-to-text, personality insights, sentiment analysis, image classification and tagging, face detection, and optical character recognition.

The creators of these systems have put a great deal of work into the development and testing of their underlying algorithms as well as the gathering of large and well-cleaned datasets that will ensure robust functionality. These turnkey features can be utilized without

any further knowledge of the underlying algorithms or datasets using straightforward API calls in an assortment of languages.

For example, an image classification query can be performed using the Node interface of the IBM Watson platform with the code used in Figure 1-22.

Example request

```
var watson = require('watson-developer-cloud');
var fs = require('fs');

var visual_recognition = watson.visual_recognition({
  username: '{username}',
  password: '{password}',
  version: 'v2-beta',
  version_date: '2015-12-02'
});

var params = {
        images_file: fs.createReadStream('./test.jpg'),
        classifier_ids: fs.readFileSync('./classifierlist.json')
};

visual_recognition.classify(params,
        function(err, response) {
         if (err)
                console.log(err);
         else
                console.log(JSON.stringify(response, null, 2));
});
```

Figure 1-22. A Watson image classification query in Node

Watson would subsequently return the JSON response in Figure 1-23, which can be easily parsed by the client system and integrated into the user-facing design:

Figure 1-23. An easily parsed JSON response to a Watson image classification query

Custom-Trained Systems

Aside from the areas of turn key functionality listed above, MLaaS platforms can be used in a wide range of machine learning problems that require custom, designer-supplied datasets. In such cases, designers may circumvent the arduous process of building, testing and deploying the machine learning system itself. They will, however, still need to devote time and resources to ensuring that the datasets they provide to these systems are clean and well-curated, though many MLaaS platforms offer substantial assistance in streamlining these processes as well.

Though the specific functionality offered will differ by platform, many MLaaS platforms provide functionality related to user behavior prediction, customer analytics and insights, inventory trends, recommendation engines, content personalization, fraud and anomaly detection, and any other supervised learning problem.

The training process for these systems generally involves the designer uploading a spreadsheet or formatted data to the platform, waiting for the model to be trained in the cloud, and then testing its behavior before deploying the functionality within a user-facing design. For most supervised learning problems, the data spreadsheet supplied to the MLaaS platform would contain at least two columns: one or more columns to represent the input attributes for a particular example and another column to represent the desired output

associated with that input. As with any training process, a larger number of examples (or spreadsheet rows) is likely to result in a more robust model. Once trained, queries to the model are performed in a similar manner to the example shown above for turnkey features.

Open Source Machine Learning Toolkits

For some machine learning problems and user-facing platforms, particularly native applications designed for offline usage, it may be necessary to deploy technologies outside of the MLaaS platforms described above. In such instances, designers may avoid at least some aspects of the lengthy development processes associated with fully customized solutions by building on top of one of a growing list of open source machine learning toolkits, which are available for a variety of programming languages and platforms. Some popular toolkits of this kind include: TensorFlow, Torch, Caffe, cuDNN, Theano, Scikit-learn, Shogun, Spark MLlib, and Deeplearning4j.

These lower-level toolkits generally require more programming experience than is necessary for the use of MLaaS platforms. Additionally, deeper knowledge of specific machine learning algorithms and their associated training techniques will likely be required for their effective use. While the MLaaS platforms listed above tend to include automated training processes, these toolkits will require designers to tune algorithmic hyperparameters such as the learning rate to align with the specific characteristics of their chosen dataset. This tuning process is aided by a deeper knowledge of mathematics and often involves at least some time-intensive trial-and-error to find suitable values.

Training a machine learning system tends to be a highly computationally intensive process, and for large or complex datasets, it is often impractical or even impossible to perform this process on a single consumer-grade machine. Many of the toolkits mentioned above are designed for use on large-scale hardware systems that use numerous CPUs or high-performance GPUs to perform training. This hardware can be cost-prohibitive and may require specialized knowledge related to the performance-enhancement mechanisms employed by a specific toolkit.

Once trained, the system will still need to be deployed to the desired user-facing platform. In most instances for which these toolkits are

applicable, the user-facing platform will have little in common with the large-scale system utilized during the training process. This means that designers must navigate two separate sets of technical and infrastructural challenges in developing their systems and making them available to users.

Despite these challenges, these toolkits offer a viable pathway for designers wishing to add customized machine learning functionality to user-facing systems. Many of these tools are backed by large technology companies, who have a vested interest in their wider adoption and are working to progressively make their tools more accessible to a wider audience of designers and developers.

One additional selling point of these toolkits over MLaaS platforms is that the tools themselves are free to use and deploy, though training them on a large-scale platform may require designers to either acquire their own costly hardware or pay for the use of a cloud-based system.

Like MLaaS platforms, the trained models developed using a toolkit of this kind may not be easily transferred for use with a different toolkit. However, in most cases, these customizable toolkits provide a more straightforward path for doing so than would be possible for a system backed by one of the MLaaS platforms.

Fully Customized Machine Learning Tools

The open source toolkits listed above strive to provide thoroughly tested implementations of proven machine learning algorithms and techniques. As a result, these tools may not include more experimental algorithms coming from recent research that has not yet been thoroughly reviewed and field tested. In some instances, these advances may provide incremental improvements to existing algorithms and in other cases may offer revolutionary new functionality. Integrating these technologies into user-facing systems will almost always require custom implementation work involving the translation of algorithms from their mathematical notation in formal research papers to working code. Rigorous testing as well as additional implementation work related to the performance and scaling of the system would also likely be required. This work generally requires a large team of developers with advanced knowledge of theoretical machine learning as well as deployment technologies. For these reasons, this approach is not advisable in most instances.

Designers who are interested in working with experimental machine learning technologies should consider joining larger teams, which are better suited to the multifaceted task of making these emerging technologies ready for production.

Machine Learning Prototyping Tools

Prototyping is an essential component of many design processes. It helps designers to sketch the basic functioning of their systems, test assumptions about how users will interact with planned features, and fine-tune those features before turning to more costly and time-intensive implementation processes. Unfortunately, the complex architectures and computationally intensive training processes associated with machine learning present challenges for the rapid prototyping of machine-learning-enhanced systems. In cases where a turnkey MLaaS solution is applicable, designers may be able to prototype their ideas with relative ease. But in instances where the desired functionality requires a custom dataset or code, producing even a basic prototype can take substantial time, effort, and know-how. Presently, there are a limited number of tools available for assisting the prototyping of machine learning systems—a circumstance that will hopefully change over the next few years.

In the meantime, several existing tools may help to ease the prototyping process. From the programming-free Wekinator to the Mathematica interface and the more programming-intensive Keras, the series of tools presented below can serve as stepping stones for students and designers wishing to prototype solutions to real-world design problems while becoming acquainted with machine learning systems and workflows in a hands-on way.

Wekinator

Wekinator is a free, open source tool created by Rebecca Fiebrink. It allows users to develop experimental gesture recognition systems and interface controllers for a wide range of input devices including webcams, microphones, game controllers, Kinect, Leap Motion, physical actuators (through an Arduino), keyboards, and mice. Unlike many machine learning workflows, Wekinator requires no programming or wrangling of datasets. Instead, the software walks the designer through an interactive process in which she defines a particular gesture by demonstrating it to the machine and then associates the gesture with a desired output action. To aid prototyping

and integration with other software, a trained Wekinator model can be set to broadcast its output events using the popular OSC protocol. Though Wekinator is only geared toward a specific domain of machine learning functionality, it provides a powerful medium for prototyping and designing machine-learning-enhanced multimodal user interfaces. See Figure 1-24.

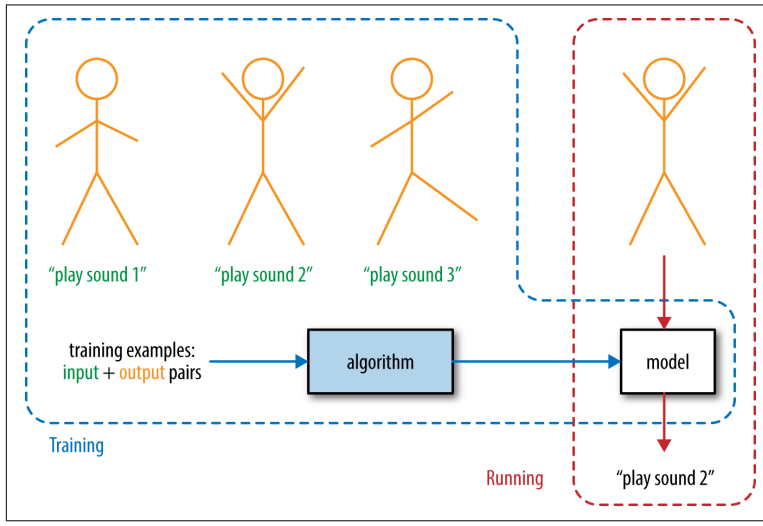


Figure 1-24. A workflow for designing event triggers with Wekinator

Wekinator can be downloaded from: *http://www.wekinator.org*.

Mathematica

The popular technical computing platform Mathematica has added a wide range of automated machine learning features to its most recent version, Mathematica 10. This tool features a polished user interface and does not require a deep understanding of programming, though some basic familiarity with text-based scripting will be helpful to new users. Its machine learning features can be applied to a wide range of data types and its automatic data preprocessing and model selection features will help users to get good results without a great deal of trial-and-error or deep knowledge of a particular model's training parameters. Mathematica provides turnkey support for a range of common machine learning tasks such as image recognition, text classification, and classification or regression of generic data. Datasets can be loaded through an interactive, visual interface. Mathematica is extremely well documented and embeds assistive

tools like feature suggestion and autocompletion directly into its interface. See Figure 1-25.

Figure 1-25. Classifying handwritten digits in Mathematica

Mathematical can be purchased from *https://www.wolfram.com/mathematica*.

Keras

This tool requires a deeper knowledge of programming as well as familiarity with command-line-based installation processes, but provides a relatively user-friendly wrapper for the high-performance machine learning toolkits TensorFlow and Theano. Though it requires programming, Keras is geared towards the rapid prototyping of highly customized machine learning systems. It provides a high-level API and modular components that will help users to

assemble common machine learning architectures such as convolutional and recurrent neural networks. This tool is not intended for new users, but may help to bridge the gap between the tools listed above and more open-ended platforms such as those listed in the following section, "Open Source Machine Learning Toolkits" on page 70.

Keras installation instructions can be found at *http://keras.io*.

Incorporating Machine Learning into Design Processes

In thinking about how to teach a person a complex task, it can be difficult to break the task down into a series of well-defined, discrete steps. The same problem can arise when designing machine-learning-enhanced systems. We might think, "the intelligent assistant should read the user's emotional state and respond accordingly." But what does this mean? Though the answer may seem relatively clear in human terms, it is less so in computational ones. Is the user's emotional state defined by his word choice? If so, how are we deciding what words are correlated with which emotions?

The first step in bringing machine learning to a design project should always be to try to define the learning problem as clearly and fully as possible. What are the input parameters we will provide the machine? What kinds of outputs are we looking for? What kind of training data will exemplify the correlations between these inputs and outputs?

Once you feel confident that the learning problem has been well-defined, it may be helpful to set aside the technical details for a moment and treat the machine learning component as a black box function within the overall architecture of your system. This means that, much like a mathematical operation such as a square root, you know what the input and corresponding output should be, but you do not necessarily know how the square root is actually computed within this function. In this way, as you sketch out the user and data flows of your system, a machine learning component can be treated like any other feature of the software. An image recognition component, for instance, could be thought of as a box that takes an image as input and outputs a list of words. This approach allows designers to incorporate machine learning features into their systems without getting bogged down in technical details during the important ideation and sketching stages, which often require fluid thinking.

Unlike a simple mathematical function, however, it is much more difficult to be certain that a machine learning feature will do what you need it to or that you will be able to find the appropriate training data to achieve the desired behavior. Therefore, this sketching process should be treated with care. If a machine-learning-enhanced feature is critical to the software's overall behavior and its efficacy is in doubt, it will be important to prototype the black box functionality to test assumptions before getting too far into designing other features around this machine learning functionality. The prototyping tools listed in the previous section may provide some assistance in getting a sense of whether the functionality will be achievable.

This process of prototyping and validating assumptions can be quite labor intensive. It may require the procurement and cleaning of a dataset, the selection of a machine learning model and a lengthy training process to see any preliminary results whatsoever. Over time, however, as you work more with machine learning systems, you will develop an intuition for what is likely to work and what kinds of learning problems may be more touchy or brittle. It is important to jump in and get your hands dirty—getting as much first-hand experience as possible is crucial. Collaboration is also of great importance. If you are working with machine learning engineers, try to form your own opinion of whether a particular idea will work and then ask for the engineer's opinion. If her opinion conflicts with yours, ask questions. Which of your assumptions were faulty? What factors did you not consider? Machine learning may be a rigorous science, but it is still something for which you can build an intuition.

As you work toward this intuition, start from simpler mechanisms and build towards more complex ones. It will not be easy to intuit how a Jeopardy-playing AI might be constructed, for example. In truth, IBM's Watson is not comprised of one machine learning system—rather, it is many interconnected components. As you introduce machine learning features into your designs, think about them as individual components. If you cannot reason clearly about what a particular component should do and what data it should be trained on, then most likely the machine won't be able to figure this out either.

Learning is an abstract phenomenon, but its role within an individual component of a design need not be abstract. In any design process, it's necessary to think back and forth between the high-level

purpose of a feature and its specific technical constraints in order to balance the many interrelated properties of a complex system. For machine-learning-enhanced features, finding this balance can be difficult. But designers can meet this challenge if they are willing to experiment, question their own thinking, and in so doing, continually strengthen their intuition for the essence of machine learning.

Conclusions

In many ways, machine learning is a solution in search of a problem. Machine learning algorithms are capable of discovering complex patterns in the data presented to them, but they are only useful if they have been trained to notice something useful. For some fields, such as finance and medicine, there are clear connections between the field's existing needs and the capabilities of machine learning systems. Financial institutions have always had a need for tools that help to predict the future behaviors of markets based on their past performance. Medical institutions have always had a need for tools that can predict patient outcomes. Machine learning simply provides more effective mechanisms for achieving these goals.

In the coming years, countless other fields will be transformed by machine learning. In many cases, however, this transformation will not be about connecting existing goals with new mechanisms for achieving them. It will require the discovery of new premises and mindsets—ones that expose entirely new opportunities and goals that can only be seen through the perspective of machine learning.

In his book *Operating Manual for Spaceship Earth*, the visionary designer Buckminster Fuller wrote, "If you are in a shipwreck and all the boats are gone, a piano top buoyant enough to keep you afloat that comes along makes a fortuitous life preserver. But this is not to say that the best way to design a life preserver is in the form of a piano top. I think that we are clinging to a great many piano tops in accepting yesterday's fortuitous contrivings as constituting the only means for solving a given problem."[11]

In looking at the history of digital design tools themselves, we may see countless piano tops. Many of the features offered by video edit-

11 R. Buckminster Fuller, Operating Manual for Spaceship Earth (Carbondale, IL: Southern Illinois University Press, 1969).

ing software, for instance, reference the preceding vocabulary of flatbed film editors. Though these references were helpful in transitioning a generation of filmmakers to a digital workflow, they did little to uncover new possibilities within the emerging medium of video. Discovering the unique possibilities of a medium requires experimentation, a fresh pair of eyes, and a willingness to think outside of the existing paradigms. It is here that designers will prove essential to the future of machine learning.

In order to fully capitalize on the technical possibilities of machine learning systems, designers will be somewhat reliant upon programmers. But programmers must also rely upon designers to find groundbreaking applications and ways of thinking about these general-purpose tools. To facilitate collaboration with programmers and develop novel applications, designers do not necessarily need to understand all of the mathematical details associated with machine learning techniques. Still, to think freely and inventively about the possibilities of a medium, it is important to understand its underlying properties and constraints. As Bob Dylan said, "to live outside the law, you have to be honest." In other words, you have to understand the rules to know which are worth bending or breaking.

To that end, for the field of machine learning to expand and thrive into the future, it will be essential for designers to immerse themselves in the possibilities of this technology, transforming it through their ways of seeing and thinking about the world.

Going Further

Staying Up-to-date with Advancements in the Field

arXiv

arXiv (pronounced "archive") is a repository of prepress scientific papers. Many cutting-edge advancements in the field of machine learning are posted to arXiv first. Keeping an eye on the latest papers posted to arXiv is one of the best ways to keep up with the latest advancements. But, with thousands of papers in a wide range of field posted to the site each month, finding papers relevant to your specific interests is not always easy.

arXiv Machine Learning: *http://arxiv.org/list/stat.ML/recent*

arXiv Neural and Evolutionary Computing: *http://arxiv.org/list/cs.NE/recent*

arXiv Artificial Intelligence: *http://arxiv.org/list/cs.AI/recent*

CreativeAI

Finding relevant papers on arXiv can be challenging. The site CreativeAI curates a collection of machine learning projects that are directly relevant to design and the arts. The projects featured on this site include written papers, videos, and even code samples. CreativeAI highlights some of the many inspirational possibilities for incorporating machine learning into creative applications.

CreativeAI: *http://www.creativeai.net*

Reddit

Another great way to keep track of important advancements that have been posted to arXiv and other sources is to keep an eye on the conversations happening within the Machine Learning and Artificial Intelligence sections of the Reddit discussion forum. Readers will find links to recently published articles as well as a wide range of discussions on topics that will be of interest to any machine learning researcher or designer.

Reddit Machine Learning: *https://www.reddit.com/r/machinelearning*

Reddit Artificial Intelligence: *https://www.reddit.com/r/artificial*

Deep Learning News & Hacker News

The recently established Deep Learning News site offers topical discussions on machine learning in a similar vein to the Reddit forums discussed above. The more general purpose Hacker News discussion forum also provides many relevant conversations about state-of-the-art machine learning technologies.

Deep Learning News: *http://news.startup.ml*

Hacker News: *https://news.ycombinator.com*

Resources for Further Study of Machine Learning

Online Courses

"Machine Learning for Musicians and Artists" taught by Rebecca Fiebrink:

https://www.kadenze.com/courses/machine-learning-for-musicians-and-artists/info

"Machine Learning" taught by Andrew Ng:

https://www.coursera.org/learn/machine-learning

"Neural Networks for Machine Learning" taught by Geoffrey Hinton:

https://www.coursera.org/course/neuralnets

Math for Machine Learning

"Some Basic Mathematics for Machine Learning" by Iain Murray and Angela J. Yu:

http://www.cogsci.ucsd.edu/~ajyu/Teaching/Cogs118A_wi10/Refs/basic_math.pdf

"Math for Machine Learning" by Hal Daumé III:

http://www.umiacs.umd.edu/~hal/courses/2013S_ML/math4ml.pdf

"Machine Learning Math Essentials Part I & II" by Jeff Howbert:

http://courses.washington.edu/css490/2012.Winter/lecture_slides/02_math_essentials.pdf

http://courses.washington.edu/css490/2012.Winter/lecture_slides/06a_math_essentials_2.pdf

"Immersive Linear Algebra" by J. Ström, K. Åström, and T. Akenine-Möller:

http://immersivemath.com/ila/index.html

"Linear Algebra" by Khan Academy:

https://www.khanacademy.org/math/linear-algebra

"Probability and Statistics" by Khan Academy:

https://www.khanacademy.org/math/probability

"Differential Calculus" by Khan Academy:

https://www.khanacademy.org/math/differential-calculus

Tutorials

"Deep Learning Tutorials":

http://deeplearning.net/reading-list/tutorials

"A Deep Learning Tutorial: From Perceptrons to Deep Networks":

https://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks

"Deep Learning From the Bottom Up":

https://www.metacademy.org/roadmaps/rgrosse/deep_learning

Technical Resources

Machine-Learning-as-a-Service Platforms

IBM Watson: *http://www.ibm.com/smarterplanet/us/en/ibmwatson*

Amazon Machine Learning: *https://aws.amazon.com/machine-learning*

Google Prediction API: *https://cloud.google.com/prediction*

Microsoft Azure: *https://azure.microsoft.com/en-us/services/machine-learning*

BigML: *https://bigml.com*

ClarifAI: *https://www.clarifai.com/*

Open Source Machine Learning Toolkits

TensorFlow (C++, Python): *https://www.tensorflow.org*

Torch (C, Lua): *http://torch.ch*

Caffe (C++): *http://caffe.berkeleyvision.org*

cuDNN (C++, CUDA): *https://developer.nvidia.com/cudnn*

Theano (Python): *http://deeplearning.net/software/theano*

Scikit-learn (Python): *http://scikit-learn.org*

Shogun (C++, Python, Java, Lua, others): *http://www.shogun-toolbox.org*

Spark MLlib (Python, Java, Scala): *http://spark.apache.org/mllib*

Deeplearning4j (Java, Scala): *http://deeplearning4j.org*

Datasets

UCI Machine Learning Repository: *http://archive.ics.uci.edu/ml*

MNIST Database of Handwritten Digits: *http://yann.lecun.com/exdb/mnist*

CIFAR Labeled Image Datasets: *http://www.cs.toronto.edu/~kriz/cifar.html*

ImageNet Image Database: *http://www.image-net.org*

Microsoft Common Objects in Context: *http://mscoco.org/home*

About the Author

Patrick Hebron is a Scientist-in-Residence and Adjunct Graduate Professor at NYU's Interactive Telecommunications Program. His research relates to the development of machine-learning-enhanced digital design tools. He is the creator of Foil, a next-generation design and programming environment that aims to extend the creative reach of its user through the assistive capacities of machine learning. Patrick has worked as a software developer and design consultant for numerous corporate and cultural institution clients including Google, Oracle, Guggenheim/BMW Labs and the Edward M. Kennedy Institute.

Acknowledgements

For Rue and our little learning machine, Lucian, whose loving support and brilliant guidance made this project possible.