A Feeling for the Algorithm: Working Knowledge and Big Data in Biology

by Hallam Stevens*

ABSTRACT

The term “Big Data” may serve as a useful marker for particular kinds of questions, practices, and relationships for collecting and using data. Some of the ways of talking about Big Data suggest that there might be something, if not entirely new, then at least importantly different at work in Big Data practices and problem-solving approaches. By using three examples taken from the biomedical sciences—artificial neural networks, the construction of reference genomes, and the usage of the Ensembl database—this essay shows how Big Data practices cannot be understood as mere scaling up of pen-and-paper methods but constitute qualitatively different kinds of knowledge-making practices. These practices are characterized particularly by types of human-computer interaction that are labeled “a feeling for the algorithm.”

INTRODUCTION

Historians are ordinarily skeptical of any claims to newness. No doubt some of the exceptionalism claimed by proponents of “Big Data” is a product of marketing and hype. Nevertheless, Big Data may serve as a useful marker for particular kinds of questions, practices, and relationships for collecting and using data. Some of the ways of talking about Big Data described in the introduction to this volume suggest that there might be something—if not entirely new—then at least importantly different at work. The connections between older and newer ways of using data may not be straightforward, and the origins of Big Data practices may turn out to be quite independent of the kinds of practices and tools that have been applied to “small data.”

The practices of Big Data, of course, are computational. Big Data are inseparable from the computers, databases, hardware, and software, in and through which they

* History Programme, School of Humanities, Nanyang Technological University, 14 Nanyang Drive, Singapore 637332; hstevens@ntu.edu.sg.
I would like to acknowledge the valuable assistance of the volume editors, Elena Aronova, Christine von Oertzen, and David Sepkoski, as well as the Osiris general editors and the anonymous reviewers. Work on this essay was supported by the Max Planck Institute for the History of Science (Berlin) as well as a Tier 1 Grant from the Ministry of Education (Singapore; RG56/13).

© 2017 by The History of Science Society. All rights reserved. 0369-7827/11/2017-0008$10.00

OSIRIS 2017, 32 : 151–174 151
are collected, analyzed, stored, sampled, shared, and displayed. As such, these technologies exert an important influence (although not a fully determinative one) on what Big Data are and what they can be and do. Looking at the history of these technologies is going to be critical for finding a history of Big Data.

Historians have been wary about attributing an epistemically transformative role to information technology in the sciences. Jon Agar, for instance, argues that early practices involving digital electronic computers essentially reproduced what scientists had been doing offline. Sabina Leonelli is also cautious in her claim that Big Data does have “specific epistemic and methodological characteristics.” For some fields, Leonelli argues, Big Data gives a new prominence to data as a commodity and entails the “emergence of a new set of methods, infrastructures and skills.” But Leonelli has little more to say about what these methods, infrastructures, and skills actually are. In this volume, David Sepkoski’s genealogy of “paper databases” suggests continuity between nineteenth-century “book” work and twentieth-century computerized practices in the field of paleobiology, and Judith Kaplan demonstrates the persistence of basic vocabulary lists in linguistics.

The aim of this essay is not to deny that some such continuities do exist, but rather to show more specifically what methods, infrastructures, and skills are associated with Big Data and to suggest how they are different. This applies first and foremost to recent usages of data (from the 1980s onward). Agar’s argument applies to the first generation of digital electronic computers and the first generation of scientists to use them. But much has changed in computing since the 1950s and 1960s. It is perhaps not surprising that the first scientists to use computers imagined few entirely novel uses for them. However, as computers became more powerful—and users realized their potential—different uses emerged. This happened only gradually, and certainly at different times in different disciplines. Over time, quantitative differences become qualitatively important (e.g., differences in speed). Although scientists may have initially reproduced paper practices with computers (and sometimes still do), the ability to massively speed up some kinds of operations does make a difference to the types of questions scientists would ask and the types of solutions they could produce.

To put this another way, the aim of this essay is not to draw a sharp line between computational and noncomputational practices. Of course, many kinds of noncom-


5 Patrick McCray, for instance, documents changing practices and norms associated with the “born digital” era in observational astronomy—a field that has dealt with large data volumes for a long time. See McCray, “The Biggest Data of All: Making and Sharing a Digital Universe,” in this volume.
putational practices can be reproduced on a computer, and many kinds of digital electronic operations can be reproduced without computers. Rather, the aim here is to distinguish some kinds of computational practices from others. "Big Data" may be a useful label for beginning to distinguish certain kinds of computational practices that are not merely reproducing paper-based practices and that rely on new ways of thinking and working. This argument does take explicit aim at the notion that we should write a "long" and continuous history of data practices that stretches back before the twentieth century. There may be, as Krajewski argues in this volume, some continuities between information processing in the Library of Alexandria and in modern online public access catalog (OPAC) library systems. But putting all the world’s books into Google’s database, for example, opens up new possibilities for manipulating and analyzing information that were not available in the time of Callimachus.

Data-inside-computers can take on different shapes and interact with itself and with us in different ways. Information collected for science is often put inside of computers, but this is frequently done so that it can take on distinct forms and be manipulated in ways that are not possible without a computer. "The kinds of machines we use are bound up with the ways we think about nature and the ways we know it," John Tresch argues at the start of his book about nineteenth-century machines. "When our machines and our understanding of them change, so does nature, and so does our view of knowledge." This must certainly be true of the digital electronic computer, perhaps more than any other machine. In what kinds of ways has the computer affected our understanding of nature? In what ways have computers reorganized the relationship between people and machines?

Here I want to examine one set of ways in which computers influence our understanding of nature by fostering particular forms of practice for producing knowledge. These practices, which involve complex interactions between human users and computers, are based on what might be called "a feeling for the algorithm." According to Evelyn Fox Keller’s account, the success of Barbara McClintock’s approach to biology relied on a "feeling for the organism." "Organisms have a life and order of their own that scientists can only partly fathom," McClintock argued. Understanding them requires the patience to "hear what the material has to say to you . . . how it grows, understand its parts, understand when something is going wrong with it." Above all, a "feeling for the organism" requires an appreciation for its complexity and a capacity to be surprised by it.

Algorithms are not (yet) as complex as organisms, so the analogy is no doubt an imperfect one. But I argue that producing knowledge with algorithms requires the same sorts of intuition, tacit knowledge, and close attention required in working with organisms. I will use the term "algorithm" here to mean not merely the high-level or

---

6 The line I am attempting to draw does not correspond to the line between, say, mechanical or electromechanical computers and digital electronic computers. Rather, it attempts to distinguish some kinds of digital electronic practices from others.
step-by-step description of a procedure for solving a problem, but rather a fully implemented, working version of a program that can actually run on a computer. In other words, the algorithm is not just its description but rather involves a set of practices through which users can interact with both machines and data in specific ways.

This essay will develop three examples—all taken from the biomedical sciences—that suggest more specifically how Big Data is shaped by computers and computing practices. That is, computers (and computational practices) have generated specific ways of thinking and specific kinds of problem-solving approaches and practices that now fall under the label of “Big Data.” To make this claim more precise (and to suggest how exactly data-inside-computers might be different), the essay will begin by introducing Davis Baird’s notion of “thing knowledge.”11 Baird’s claim that scientific instruments embody a distinct form of knowledge can be usefully appropriated for computing to show how it can involve new forms of knowledge and knowledge making. The aim of these examples is to suggest that algorithms and data structures contain various forms of “working knowledge” that are important in the making of new biological knowledge. In other words, these algorithms and data structures do not merely store, manage, and circulate biological data; rather, they play an active role in shaping it into knowledge.

THING KNOWLEDGE, WORKING KNOWLEDGE, AND COMPUTERS

Baird argues that scientific instruments can, in themselves, constitute various forms of scientific knowledge. This can occur, he argues, in three ways. First, scientific instruments may serve as models of the world or of various phenomena. One of the examples Baird gives is the orrery: as a model of the motions of the planets around the earth, the device embodies a kind of knowledge of the heavens. Second, Baird claims that instruments can contain something called “working knowledge.” In this case, he means a device that is not intended as a model of a phenomenon but that demonstrates or instantiates a phenomenon without necessarily providing an understanding or explanation of it. A superconducting magnet, for example, possesses a “working knowledge” of superconduction, even in the absence of any theory to explain how or why it does what it does. Furthermore, tinkering with the device and improving it may lead us not only to better devices, but also to a greater understanding of what is going on. Third, there are measuring instruments. For Baird, these are a kind of hybrid of models and working knowledge.

Computers have been used in certainly the first and third ways in recent science: it is possible to use the computer to make simulations or models (e.g., of protein folding). And it is possible, by hooking a computer up to appropriate apparatus, to make it into a measuring instrument. Here, however, I will be concerned with the second category: that is, of machines instantiating “working knowledge” of some phenomenon. According to Baird a machine itself can contain a certain kind of knowledge about the world that is not the same as a picture or words or a set of equations representing or describing it. A working superconducting magnet, again, contains a particular kind of knowledge about superconduction. That is, we can learn about the phenomenon of superconduction by building and using a working device. As Baird noted, “We

11 Davis Baird, Thing Knowledge: A Philosophy of Scientific Instruments (Berkeley and Los Angeles, 2004).
say someone knows how to ride a bicycle when he or she can consistently and successfully accomplish the task.”12 Likewise, knowing how to build and use a superconducting magnet means we have a “know-how” or “skill knowledge” with respect to superconduction. This knowledge does not depend, Baird argues, on any theory about superconduction or any explanation of how the phenomenon works. “We learn by interacting with bits of the world even when our words for how these bits work are inadequate,” Baird claims.13

A successful working device, Baird continues, must have a kind of “cognitive autonomy.” There is something epistemologically important about a working machine that literary (or mathematical) description misses. It is possible to learn things from machines, or to refine them, not by mathematical reasoning or textual description, but rather by manipulation of the device itself. Machines have a distinct “cognitive channel” that allows new knowledge to be created while remaining “ignorant of theory” and even while remaining unable to express in words either what the instrument teaches you or what you are doing with the knowledge so gained.14

In sum, there are two crucial aspects of Baird’s “working knowledge.” First, through a device, machine, or instrument, it is possible to do something, or make something work, without understanding in linguistic or descriptive terms everything that the device or machine is doing. And second, by using and manipulating devices or instruments it is possible to generate new knowledge, new ideas, or new machines without understanding fully how that device works or what it is doing.15

Baird does not discuss computers. This is no doubt because they present a complicating case for his argument. At the center of Baird’s claim is an argument for paying attention to, and taking seriously, materiality and material manipulations. However, many things that are important and distinct about computers are not material. Computers, of course, deal with the “virtual”: they virtually manipulate language, numbers, and equations. In this sense it is obvious computers are doing things with knowledge—they actually have linguistic and numerical and mathematical knowledge encoded into them. On these terms it seems like computers are simply behaving as an inscription device—they are allowing us another way to write things down that is not fundamentally different from other written forms of knowledge. This corresponds to the idea that computers are doing the same things we can do on paper, only faster or on a larger scale.

This reduction of knowledge to the linguistic is exactly what Baird is attempting to resist. However, I want to suggest that there is something going on with computers beyond this mundane sense of them as sophisticated inscription devices. There are nonlinguistic ways in which computers are routinely used that are very much like Baird’s descriptions of “working knowledge.” First, algorithms or software do things,
make things, and behave in ways that are not always predictable or intended or understandable. Sufficiently complex algorithms (especially working with sufficiently large data sets) exhibit the kinds of behavior that seem similar to the cognitive autonomy Baird describes. In other words, software can do things that are often not reducible to theory or words. In working with simulations, fractals, artificial life, or artificial intelligence, the aim is exactly to produce programs that exhibit behaviors that are not readily predictable from the code on which they are based. Where large amounts of data are involved as well, data structures and databases can organize information in ways that may generate surprising results. The aim of Big Data is to organize data in particular ways that allow us to see patterns that would otherwise be impossible to discern.

The idea that computers embody working knowledge is supported by recent work in the history of mathematics. Stephanie Dick has explored how mathematicians at Argonne National Laboratory in the 1970s and 1980s deployed software called the Automated Reasoning Assistant (AURA) to help them search for mathematical proofs. Dick details how the computer successfully produced proofs via complex interactions and “negotiations” between the human mathematicians and their machine assistant:

AURA was full of surprises. Although the program could only do precisely what it was programmed to do, the programmers usually could not know in advance what the consequences of their instructions would be. Indeed, if they could, they would have had little need for a fast computer to do this work for them. They manually studied printouts of thousands of clauses from run after run on AURA in order to understand what was going on when the engine looked for proofs. To accommodate their a posteriori revelations, the Argonne group committed to an experimental paradigm for ATP in which AURA was constantly improved and redesigned on the basis of results from previous runs. Those prized human insights and intuitions that the Argonne researchers reserved for the human, rather than materializing from the cognitive ether, emerged from intimate and prolonged experimental work with the engine. Experimentation determined the character, form, and relevance of human contributions to AURA proof searches. By privileging and isolating a traditional notion of human mathematical thought in their design, the Argonne team in fact made possible radically new forms of intuition and insight grounded in experiential knowledge of computational behavior.

Working with AURA involved a new kind of work in which the human and the machine should be considered a kind of hybrid. This kind of “experimental work” with algorithms is not confined to mathematics. Rather, this kind of interaction between computers and humans is a feature of scientific computing more generally.

One way of describing a complex engineered system—such as a computer, an algorithm, or a piece of software—is in terms of black-boxing. A software user may not

---


know much about how that software works; even a software designer may not know the detailed structure of a subroutine that he or she codes into a finished program. This ignorance would not stop the user (or the coder) from successfully using the software (or the subroutine) as an input-output device. In many cases, such ignorance may actually be advantageous for successful usage. What I am attempting to capture here is the opposite of simple black-boxing. Black-boxing usually refers to situations in which, in principle, the box can be opened up. What is inside the box may be hard to find out, unimportant for a given purpose, or protected by trade secrecy or some other means. Nevertheless, one can coherently imagine that someone knows what is inside the box.

On the other hand, saying that algorithms and data structures can contain working knowledge is to suggest more fundamental kinds of ignorance. It is not simply that a user or coder does not know how a program works. Rather, the user or coder cannot have access to the important knowledge embodied in and produced by interaction with the computer. The user may know in general or descriptive terms how a program works. But this is not the same as the working knowledge that can be produced through interaction with the machine. In Baird’s terms, this sort of knowledge occupies a “cognitive channel” distinct from a description of the algorithm. The only way to obtain the knowledge produced by an algorithm is actually to use it.\(^{19}\) We can observe the detailed properties of an algorithm (the lines of code), but this does not mean we can predict the outcome the algorithm will produce.

Even after we have run an algorithm it may be very difficult to understand precisely how it arrived at a given result. This ignorance might arise from stochastic elements of the program, from nonlinearities in its mathematics, from the complexity of the operations it performs, from the size and/or multidimensionality of the data sets with which the algorithm is working, or from some combination of these factors. This makes a significant difference to how we can use such algorithms—we use them by playing with them, running them, tweaking them, interacting with them, and so on. Through these sorts of practices we can come to develop a “feel” for their detailed behavior. It is in this sense that using algorithms seems very much like using other kinds of instruments and objects in the ways described by Baird. There is something about the complexity of algorithms and data structures that makes them more like the material instruments that Baird discusses. In the remainder of the essay, I will develop three examples that demonstrate how algorithms and data structures are put to work in biology in this way.

**ARTIFICIAL NEURAL NETWORKS**

Artificial neural networks (ANNs) are a form of machine learning algorithm used in many different kinds of applications. Building on the work of Warren McCulloch and Walter Pitts in the 1940s, simple ANNs were designed by early computer architects

\(^{19}\) This has some features in common with the concept of emergence in physics, where “the micro-level interactions are interwoven in such a complicated network that the global behavior has no simple explanation.” See Mark Bedau, “Downward Causation and Autonomy in Weak Emergence,” in *Emergence: Contemporary Readings in Philosophy of Science*, ed. Mark A. Bedau and Paul Humphreys (Cambridge, Mass., 2008), 155–88, on 160.
(including Wesley Clark and Frank Rosenblatt) in the 1950s. The basic idea of an ANN is to use a computer to simulate the way in which neurons are connected in the brain. In short, a single neuron (call it X) may have a range of input connections from other neurons. If those input neurons are activated (“fire”), they will transmit an electrical signal to X. If enough electrical signals are transmitted to X, it will reach its “threshold” and will also be activated. X may also have multiple output connections to other neurons. Once X is activated, it will transmit an electrical signal to those downstream neurons (which may, in turn, be activated). In this way, a signal is propagated along a specific path of neurons through the brain.

ANNs model this process. Conceptually, they consist of an “input layer” (the input data), one or more “hidden layers,” and an “output layer” (the output data). Each layer consists of several nodes (which represent neurons). To start off, each node in the input layer is randomly connected to several nodes in the hidden layer (we will assume only one hidden layer for simplicity), and each node in the hidden layer is randomly connected to nodes in the output layer (see fig. 1). The network then requires a data set with which it can be “trained.” One area in which ANNs have had considerable success is in optical character recognition. Let’s say the input layer consists of sixteen boxes that can be either light or dark (see fig. 2). And let’s say the output layer consists of the ten numerals 0 through 9. In this case, we wish to train the ANN to successfully recognize each of the ten numerals from the patterns of light and dark that each would make in a 4 × 4 grid of boxes. Our training data would consist of light and dark patterns representing the various digits plus our own knowledge of the correct responses.

To train the ANN, we use drawings of various digits 0 through 9; these drawings are reduced to the pattern of light and dark boxes that becomes the data for the input layer. Each box can be either light or data, “on” or “off.” Where the box is light, the corresponding input node is activated. Because of the connections between the input layer and the hidden layer, this activation pattern in the input layer will activate certain nodes in the hidden layer, and, in turn, the nodes in the hidden layer will activate certain nodes in the output layer. What we desire here is a pattern of connections such that, when the boxes are shown a “1,” only the first output node is activated; when the sensors are shown a “2,” only the second output node is activated; and so on. If that occurred, we would say that the ANN was successfully “recognizing” each of the drawn digits.

At first, because the connections between nodes are random, we expect that particular inputs will produce random output. However, the point of the training is to gradually change the behavior of the ANN, pointing it toward the “correct” responses. This gradual change is achieved by inputting the training data one instance at a time. When a certain pattern of light signals is inputted, the output is observed. If the ANN, by chance, produces “correct” outputs, the connections that led to this output are


21 This example is often used in teaching ANNs. For more details, see Jianchang Mao and Anil K. Jain, “Artificial Neural Networks: A Tutorial,” Computer 29 (1996): 31–44. For a general introduction, see C. M. Bishop, Neural Networks for Pattern Recognition (Oxford, 1995).
Figure 1. A schematic diagram of an artificial neural network consisting of one input layer (left), one hidden layer, and one output layer (right). The thickness of the connections between the various nodes in the layers corresponds to the strength of the connection between them. As the network is “trained,” the strength of these connections is modified to produce “correct” outputs. (Image by Hallam Stevens.)
strengthened. If, on the other hand, the network produces “incorrect” outputs, those connections are weakened. In this way, the ANN is gradually disciplined toward providing “correct” responses. Eventually, the ANN can be trained to correctly “recognize” unknown input patterns.

In biology and medicine, ANNs have a variety of applications, including prediction of protein secondary structure and adverse drug reactions, gene finding, and DNA microarray analysis. The example I will describe here involves using RNA expression data (from a microarray) to classify cancerous tumors. This sort of prediction is important because tumors differ in multiple and complex ways (some responding to drugs and others not), and it is important not to give a patient potent, toxic drugs if their particular tumor will not respond. Here the training data consist of microarray

---

22 This “updating” of the network is actually the real trick to making ANNs work. The method is called “backpropagation” and is the most mathematically involved aspect of ANNs.

23 For a sense of the range of applications of ANNs in biology and medicine, see H. Malmgren, Magnus Borga, and L. Niklasson, eds., Artificial Neural Networks in Biology and Medicine, Proceedings of the ANIMAB-1 Conference, Göteborg, Sweden, 13–16 May 2000 (New York, 2000); David J. Livingstone, Artificial Neural Networks: Methods and Applications, Methods in Molecular Biology (New York, 2009).

data that provides information about which genes in each tumor are turned “on” and “off” (this is called an “expression profile”) as well as information about the true classification of each tumor. The ANN is trained on this data in the hope that it will be able to predict the proper classification of tumors in unknown cases. Such an approach has had considerable predictive success.

In 2001, Javed Khan and colleagues attempted to use an ANN to recognize “small, round blue-cell tumors” and classify them as neuroblastomas, rhabdomyosarcomas, non-Hodgkin lymphomas, or Ewing-family tumors.25 Such tumors are difficult to distinguish via traditional methods (light microscopy), but patient prognosis differs markedly depending on diagnosis. The experiments began with sixty-three training samples (consisting of tumor tissue and tumor cell lines) and DNA microarrays containing 6,567 genes. These 6,567 inputs for each sample were reduced by principal component analysis to ten dimensions, which were used as inputs for the ANN. The outputs were the four types of tumors. The sixty-three training samples were randomly partitioned into three groups, two for training and one for validation. The training group samples were each run through the ANN one hundred times. The samples were then repartitioned and the process repeated, producing 3,750 distinct ANN models. These models were used to create a ranking of the genes that most affected the tumor classification. The entire procedure was then repeated with only the ninety-six top-ranked genes as inputs. This ANN was then tested on twenty-five different samples, correctly classifying all of them.

The point was not to try to understand how the ANN actually arrived at its predictions. Microarrays are not considered to be successful because they provide a theoretical understanding or model of what is going on inside a tumor. As Kahn et al. acknowledged, “A potential difficulty with ANN-based pattern recognition models is elucidating causal links from the outputs to the original input data.”26 Although the procedure described here attempted to mitigate this by generating a list of the “most significant genes,” most of these genes were not previously known to be involved with cancer. This suggests that a great deal more experimental work would be needed to actually understand why particular genes were showing up as significant.27 Nevertheless, the ANN results could be used to develop a lower-cost test that would distinguish different cancer types. For instance, follow-up work could design a new microarray that would test for ninety-six genes rather than 6,567 genes. The ANN produced a practical—not theoretical or mechanistic—model for diagnostically distinguishing these cancers.

The multidimensional data of the sort that comes from microarrays usually means that we have no idea how the ANN is actually arriving at its answers. Of course, it is possible to describe in words, as I have just done, how the algorithm works, or to describe it in more formal mathematical terms. But this does not provide a full description of what it is actually doing or how or why it is arriving at specific predictions. We could also write down all the connections and weightings in the network and examine

---


26 Ibid., 677.

27 And the ANN still used ninety-six genes for distinguishing just four cancer types; this is a small number compared to the 6,567 input genes, but still far too many to suggest a clear picture of how each tumor differed from the others.
them. But in most cases (and especially in cases with multidimensional data), this would not tell us much more about why the connections are the way they are or how they are producing predictions. Usually it is not apparent how ANNs are reaching a particular solution.

The only way to obtain knowledge about tumors from an ANN is for the user to actually train and run it. It is a process of interaction between the user, the data, and the algorithm itself that produces working knowledge of cancerous tumors or some other system to which it is applied. The necessarily interactive aspects of this kind of knowledge building were illustrated in work done in 2015 at Google by the software engineers Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Google is interested in using powerful ANNs (with between ten and thirty layers) to recognize specific objects in images (this has applications in Google’s image search, for instance). “One of the challenges of neural networks is understanding what exactly goes on in each layer,” Google’s Research Blog reported. To address this, the Google engineers began to get the ANNs to “draw” output images of what they “saw.” They fed random noise to an ANN trained to find images of, for instance, starfish and observed the results. The ANNs could successfully generate images themselves (e.g., of a starfish). To experiment further, the engineers began to mix things up, feeding images of mountains to ANNs designed to find buildings or images of clouds to ANNs designed to find animals (fig. 3). Detailed but surreal images of buildings and birds began to emerge from the pictures. Finally, in a technique they called “inception” (after the 2010 Christopher Nolan movie), the engineers fed back the ANNs’ own images into the algorithm over and over again, producing strange, dreamlike scenes.

Although all this seems merely playful, the experiments had a serious goal. The resulting images “help us understand how neural networks are able to carry out difficult classification tasks, improve network architecture, and check what the network has learned during training.” In other words, it helped the Google engineers to understand their tool by revealing more about the function of specific layers in the ANN. But the experiments also revealed things about the objects to which the algorithms were applied. In this case, engineers learned something about the essence of the images they were feeding to their algorithms (e.g., they learned that images of dumbbells were inseparable from images of muscled arms). This kind of tinkering with the algorithm allowed the users to develop a kind of working knowledge of their images. If “inception” were applied to ANNs trained on biological data, it might reveal useful working knowledge of biological systems (such as what features of tumors or gene networks are important or are active in a particular process).

ANNs can make accurate and reliable predictions about the world. And they capture things that we cannot fully reduce to linguistic descriptions. Moreover, we can certainly gain knowledge from an ANN: they can lead us to new and important knowledge about

---


30 See https://photos.google.com/share/AF1QipPX0SCi70zWilt9LnuQiatX4OUj_8EP65cTVnBmS1jnYgsGQAtieQUc1VQWdgQ?key=aVbxWjhwSzg2RjJWLWRuVFBBZENld205bUdEmhB (accessed 16 July 2016).

31 Mordvintsev, Olah, and Tyka, “Inceptionism” (cit. n. 29).
Figure 3. “Funny Animals” made by Google Research using artificial neural networks and the “inceptionism” techniques. (Reproduced under CCA 4.0 license; images by Alexander Mordvintsev, Christopher Olah, and Mike Tyka. See Mordvintsev, Olah, and Tyka, “Inceptionism” [cit. n. 29]. Images have been modified to black and white.)
cancerous tumors or other complex systems. Moreover, we can improve on the algorithm itself, refining it by using additional training data or interacting with it and tweaking the details of the network. This sort of work involves careful interaction with the algorithm. The knowledge derived here relies on tinkering and playing, as well as intuition about how the algorithm works. The point is not to wholly comprehend what the algorithm is actually doing in any given instance, but interacting with it allows us to learn from it or to improve on it.

REFERENCE GENOMES

My second example also involves a complex algorithm working on a large data set. In this case, however, working knowledge emerged in the interaction between the algorithm and its user. The term “reference genome” is what biologists usually mean when they talk about sequencing genomes. These are the finished sequences of most parts of a species’ chromosome (human, elephant, platypus, etc.). The Human Genome Project (HGP) aimed to build a reference genome for humans. Importantly, the reference genome is not derived from a single individual. Rather, it is ordinarily constructed from samples taken from many (or at least several) individuals of the same species; it thus becomes a representative example of the genome for that species. Reference genomes act as a “reference” since once a reference genome is complete it can be used as a scaffold for assembling the genome sequences of other individuals of the same species much more quickly.

Making a reference genome involves a number of steps. These steps have changed over time as sequencing technology and bioinformatic software has advanced. Because I am interested here in the algorithm that was used for the assembly of the reference genome for the HGP, I will describe the process as it was carried out around the year 2000. First, a single chromosome from the genome is broken up into many longer fragments in order to prepare it for insertion into bacteria or yeast. Sequencing technology requires millions of identical copies of a piece of DNA in order to produce a strong signal. Second, this copying was performed by splicing the DNA into bacterial or yeast DNA “vectors” and allowing those organisms to do the copying. Third, the copies of the long fragments are then extracted from the bacteria (or yeast) and broken randomly into many smaller fragments. These fragments are small enough to be sequenced by the sequencing machines (which can handle sequences of only up to 1,000 base pairs). Since the breakage is random, each of the copies may not be broken up in the same way. Next, each of these fragments is sequenced in a sequencing machine and stored in a computer database. And finally, computers are used to reconstruct the original long fragments. This is only possible because the random breaks mean that there will be overlapping segments of sequence that can be used to match up the pieces from end to end.

It is this last step that I am concerned with here. The “assembly problem” turned out to be a critical challenge for the HGP. This task involved 30,000 clones and 400,000 fragments, which together covered 88 percent of the human genome, or 2.7 billion base pairs. This has sometimes been compared to solving a jigsaw puzzle with 400,000 pieces.

32 “Finished” is a technical term in genomics that refers to agreed-upon levels of completeness and accuracy. See, e.g., https://www.genome.gov/10001812 (accessed 16 July 2016).
In fact, it is much more difficult: this is a jigsaw puzzle with 400,000 sometimes redundant, always overlapping pieces, some of which are missing and many of which are blurry around the edges.

For the HGP, this massive computational task was tackled by Jim Kent, a graduate student at the University of California, Santa Cruz. Kent had started his career as a software designer in the 1980s, founding a computer animation business. In the late 1990s, Kent decided to return to school to train as a biologist. His work initially focused on the worm genome. In mid-2000, the HGP was struggling to find a way to assemble its data into a complete genome. David Haussler, who was leading HGP’s assembly efforts, asked Kent, with his previous programming experience, to give it a try. Kent’s software, called GigAssembler, produced the first complete assembly of the human genome, just managing to beat out the efforts of a private company, Celera Genomics. The most computationally intensive step took about three days to run on a cluster of one hundred 800-MHz Pentium CPUs.\(^\text{33}\)

How did GigAssembler work? The blurriness of the data (arising from inaccuracies in sequencing) meant that it was necessary for Kent to make the maximum and best use of all the data at his disposal. The aim here was not getting the “right” answer (because the data came from no individual’s actual genome, there was not a single “right” answer) but rather getting the “best” answer. In practice, this meant using a set of heuristics to score the various ways of lining up the pieces of the genome, which required careful judgment calls about what was most important in constructing the alignment (e.g., is it better to have an error or a gap?). Researchers valued the relative weight of matches, mismatches, and tails (areas adjacent to matching areas that overlapped but did not match). Here is how Kent and Haussler describe the process:

The scoring function is crucial here. It is not unusual for the data to conflict. It is important that especially the first joins be based on the strongest matches. The current scoring function strongly favors overlaps that are unique, weakly favors overlaps that are repeat masked, strongly discourages sequence mismatches and inserts within the aligning blocks, and moderately discourages tails. Alignments below a certain threshold of the scoring function are not used to build rafts.\(^\text{34}\)

We can see how these decisions are built into the algorithm itself by examining a part of the code for the scoring function:

```c
int fragOverlapScore(struct psl *alignment) /* Return score from roughly 500 to -500 for fragment/fragment overlap based on alignment. */
{
    int milliBad; /* Misalignments in rough parts per thousand. */
```
This subroutine in the scoring code calculates how to "score" two overlapping fragments. The higher the score, the more likely these fragments will be joined in the finished alignment. The score is based on the fraction of mismatches in the overlap, the size of the overlapping segment overall, and the size of the tails on each end. Notably, the scoring function includes various integers (in bold) that place relative weights on these different factors.

There are other aspects of GigAssembler that exhibit similar features to the scoring subroutine. Building and using such software is not simply a matter of entering the data and waiting for the computer to finish its job: the software needs to be actively manipulated to make it work in each case. Determining the numbers in the scoring function requires judgment and intuition about how to make the best possible alignment. It also requires finding out what combinations of parameters are likely to "work" by accumulating experience with the program and adjusting parameters accordingly. The results of adjusting particular parameters are not predictable in advance—such work requires a feeling for the algorithm and a kind of tinkering to make it produce the best possible result (i.e., a genome with the least gaps, most matches, etc.).

35 Kent and Haussler, "GigAssembler" (cit. n. 33).
Kent’s description of the process of writing his program suggests how this was not simply a matter of running the program once and letting it spit out the answer:

And so I decided that I would just write something quick and simple that would do it. And it did do it. It was pretty quick to come up. It was harder than I thought at first. It took, I guess, it took about a week before the very first—when I started to the very first thing that worked at all. And for me that’s actually a long time. Usually when I write a piece of program, most programs will have the very first skeletal thing that will take about a day or two because I like to sort of build it so that you’ve got the skeleton first, and then you kind of layer stuff on top of it. But it’s much easier to test the program if you always have a little something working. And then sort of add a little bit more to do and a little bit more to it. And so I always try and get the first thing that’s kind of working and close very quickly. So it took longer than I thought. And then we just kept adding stuff to it.36

It was not only the fact that more data was becoming available that necessitated multiple runs of the algorithm. Rather, more and different types of data meant adjusting the algorithm, making improvements, and learning more about the behavior of the algorithm from the output it produced.37 Describing Kent’s achievement to the New York Times, Haussler emphasized the rapid pace of his work: “This program represents an amount of work that would have taken a team of 5 or 10 programmers at least six months or a year. Jim in four weeks created the GigAssembler by working night and day. . . . He had to ice his wrists at night because of the fury with which he created this extraordinarily complex piece of code.”38 This detail suggests the intensity of the interaction between computer and human required for this kind of work. Kent needed a deep and intuitive connection with the algorithm and his data that allowed him to do work that could ordinarily only be performed by a large team.

This kind of work resembles the kind of work that is done with instruments. That is, there is a kind of working knowledge in the algorithm that it is impossible to capture in a mere description of it. Even though it is literally possible to read through the algorithm itself and even to understand what it does in general terms, in practice this does not tell us everything about what the algorithm does, what it can make, or how it does it. This is true even for the programmer, Jim Kent himself: successfully making a reference genome involved a specific kind of interaction between the programmer (or user), the data, and the machine. It is not merely that Kent did not know all the details of how the algorithm would behave (as in the case of black-boxing), but rather that he cannot practically know how it will behave without actually running it and observing the results.

GigAssembler produced not only new biological knowledge but also a new biological object (a reference genome). Yet this production did not rely on necessarily un-

---


38 Wade, “Reading” (cit. n. 37).
derstanding everything the algorithm did, and it certainly did not rely on a complete accounting of how it arrived at its specific solution. One could know generally how GigAssembler weighed different kinds of alignments, and so on, but the user did not know the detailed process through which fragment A was aligned with fragment B, and fragment B with C, and so on. The user remained ignorant of the detailed process through which a reference genome was built. In principle, because a computer is a deterministic machine, it is possible that a human (or group of humans) could follow the exact instructions in the algorithm and make a reference genome themselves. But replicating the work done with GigAssembler would mean not just running the program once but running it over and over again with slightly different code and parameters each time. Even if this were conceivable, doing this work with human computers or on paper would engender a fundamentally different set of interactions between the “user” and the “program.” Actually making a reference genome requires tinkering, playing with the algorithm, adjusting parameters, running it again, and so on; it means interacting with the algorithm to get the “best” result out of it.

ENSEMBL AND HYPERTEXT

My third example is a database system rather than an algorithm. This case suggests that working knowledge can inhere not only in algorithms but also in the complex ways in which data is ordered and presented. Ensembl is the preeminent database of the European Bioinformatics Institute (EBI; based in Hinxton, Cambridgeshire).\textsuperscript{39} The project was initiated in 1999 in an effort to organize and systematize the data from the HGP. The database is most commonly accessed and used as the Ensembl Genome Browser: this is a Web interface that presents genome data in a visual format, allowing biologists to scroll along the lengths of various genomes, zooming in on various features (fig. 4). This is similar to other “Genome Browsers” maintained by the National Center for Biotechnology Information (at the National Library of Medicine in Bethesda, Md.) and the University of California, Santa Cruz.\textsuperscript{40}

Ensembl itself produces no data. Rather, it collects data from a wide variety of sources and makes it available in a common, consistent, and coherent format. Moreover, the Ensembl Genome Browser is the front end of a much more complex and interconnected system of databases that store data and allow access to it through a variety of tools (e.g., the data-mining tool BioMart). Ensembl’s history can be traced to a database to manage information for the worm genome project, ACeDB (A C. elegans Data Base).\textsuperscript{41} This database was initiated in 1989 as a joint project between Richard Durbin (Laboratory for Molecular Biology, Cambridge) and Jean Thierry-Mieg (Centre National de la Recherche Scientifique, Montpellier). Durbin and Thierry-Mieg wanted to find a way to store sequence information, physical and genetic maps, and literature citations within the same database. To do this, they realized that they required a very flexible system in order to manage not only the quantity of data but also


\textsuperscript{41} For more information on ACeDB, see http://www.acedb.org/ (accessed 16 July 2016).
its variety. They wanted to help biologists organize and manipulate data in order to find the important relationships within it.

The result was a system that was not merely a database but also a visualization system (fig. 5). ACeDB was being designed at the same time that Tim Berners-Lee was developing the tools of the future World Wide Web: HTML and HTTP. Durbin and Thierry-Mieg made use of these tools, linking their data using hypertext. ACeDB was not only solving a storage problem but also demonstrating how to make data useful and valuable. To do so, its designers had to find ways to organize and link the data in accessible and findable ways. For this purpose, hypertext proved particularly useful.42

ACeDB proved extremely successful. It was quickly picked up and used beyond the worm genome project. Both the Wellcome Trust Sanger Institute and the Genome Institute at Washington University adapted ACeDB to be used as their main database for

---

storing and organizing human genome sequences for the HGP. Beginning in 1998, Durbin realized that the HGP required an even more powerful tool for representing genomic sequence data. He put some of his graduate students to work on this problem: Ewan Birney, Tim Hubbard, and Michele Clamp developed the code for Ensembl using ACeDB as a model. The eventual name adopted for this database system—a “browser”—echoed the increasingly important Web browsers being developed by Netscape and others at the same time.43

The way that biologists use Ensembl also reflects how it is linked to the technologies of the Web. In Ensembl (or other genome browsers), users navigate by clicking through links, which allows them to call up or drill down to the specific data that they might need, generating and seeing relationships between various data. In most cases, specific Ensembl Web pages and links are dynamically generated from the database by the Ensembl code. That is, selecting certain options or clicking certain links will cause

the data to be displayed in new arrangements. This sort of work can reveal novel patterns or relationships in data that can tell us about how organisms evolve or function.

Ensembl’s programmers at the EBI give examples of how Ensembl’s Web tools can be used to find the function of a gene or to discover genomic elements likely to be regulating a gene. For the human gene MYO6, for instance, the browser first allows the user to look for similar genes (homologues) in other organisms (“Do so by clicking on the orthologues link at the left of the gene tab”). This reveals an extensively studied mouse gene called Myo6. Functions for this gene can be revealed by looking at the associated gene ontology (GO) terms: “Clicking on the mouse protein identifier ENSMUSP00000108893, then on the gene ontology link at the left shows the GO terms associated with the mouse protein.” The known functions of the mouse gene can be used as a basis for predicting the analogous function of the human gene. To predict regulatory sequences associated with the human gene, the authors suggest looking for histone modification and methylation sites near the gene:

Regulatory features are drawn alongside the human genome along with MYO6 transcripts, which are on the forward strand. Variations are also drawn and colour-coded indicating the position of the variation with respect to the gene (ie, intronic, upstream, coding). Clicking on variations opens a pop-up box with specific information, and a link to the variation tab.

Navigating around near the human gene can lead to new knowledge about what this gene does and how it is regulated. But it can also foster more tacit forms of knowledge about how regulatory elements act, what evolutionary relationships between genes look like, and so on. This is based on a visual, interactive, and hypertextual activity. It does not require knowing how the Ensembl system itself works, nor does it require understanding the structure of the underlying data. Instead, it is through hypertextual interaction with the data that it is possible to gain working knowledge of genes and genomes.

This is not simply a question of black-boxing. It is not just that the users of Ensembl do not need to know how it works. Rather, it is that no one (not even Ensembl’s designers) can describe or predict all the possible states that the data structure may produce. There is no practical way of reasoning through the large sets of data, operations, and constraints. Here, this has as much to do with the amount and multidimensionality of the data as with the complexity of the algorithms at work. The size and complexity of the data mean that data structures play a critical role in shaping the interaction between data and users; a user’s understanding of the data, his or her ability to manipulate it or work with it, and his or her “feel” for it depend on data structures.

The use of Ensembl is similar to how we use many parts of the Web. Web technologies allow us to navigate through data in specific ways. The Web imposes a structure on the data that constrains what we do with it. Tinkering with the Web (“surfing”) allows us to discover certain kinds of patterns and relationships that otherwise might not


45 Ibid.

be visible. For instance, websites such as Kayak (www.kayak.com) aggregate information from different databases (in this case, airline flight data) based on user input and display it in novel ways. This provides specific knowledge about the cheapest flights to particular destinations on particular days. But interacting with Kayak also fosters a kind of working knowledge of how to search for the cheapest flights, when the best deals can be found, what kinds of parameters to input, how flexible to make one’s search, and even perhaps a more intimate knowledge of how the arcane system of airline ticket pricing works.

The constraints and categories that are created by the Web itself constitute a structure within which we can operate on and interact with the data. Working knowledge emerges in the interaction between the data, the structure, and its users. Users are able to generate new knowledge through interacting with the data, even though they cannot precisely describe what that knowledge is or how it arises. The possibilities of the Web structure coupled with massive amounts of data make it impossible, in practice, to predict the kinds of relationships the data might reveal, except by actually interacting with it via tools such as Ensembl. The underlying data of the human genome or the Web elude linguistic or mathematical description. But data structures and tools provide ways of tinkering with data that allow users to “see” particular features.47 Using Web browsers or genome browsers allows users to develop a working knowledge of the underlying data through interaction with it.

CONCLUSIONS

The three examples here all show different ways in which large quantities of data have been utilized to make biological knowledge. Whether we call these “Big Data” or not, I have argued that the kinds of computational work involved here entail different practices and methods from older ways of working with pens, paper, and people. Some of these practices and methods predate the recent fascination with Big Data. But they are not necessarily related to or derived from conventional statistical or scientific data-processing methods. Instead, they have emerged from fields such as artificial intelligence and machine learning to play a more prominent role in a range of fields. “Big Data”—because it draws attention to size rather than practice—may be an inappropriate label for this kind of work, but the wide application of these methods does nevertheless mark an important shift.

This shift represents not merely a scaling up of work done with pen and paper or books and tables. Rather, the quantities of data processed by computers, and the algorithms needed to deal with them, make a qualitatively different kind of knowledge. In other words, this is a “more is different” argument. Philip Anderson in the 1970s made this argument for the importance of solid state physics vis-à-vis particle physics: having more molecules to deal with not only involved a scaling up but generated situations in which qualitatively new phenomena (such as superconductivity) emerged:

The behavior of large and complex aggregates of elementary particles, it turns out, is not to be understood in terms of a simple extrapolation of the properties of a few particles.

47 Software allows biologists to “see” their objects in particular ways that open it up for certain forms of understanding and manipulation. See Hallam Stevens, *Life Out of Sequence: A Data-Driven History of Bioinformatics* (Chicago, 2013), chap. 6.
Instead, at each level of complexity entirely new properties appear, and the understanding of behaviors requires research which I think is as fundamental in its nature as any other.48

This is an emergence argument: that the properties of the higher level cannot be practically derived from the properties of the lower level. By analogy, I am suggesting that large amounts of data, subjected to complex algorithms and data structures, present a similar situation. The behavior of such systems cannot be predicted in advance, and the objects and solutions that they produce are not reducible in any simple way to paper-based manipulations of text or numbers. At some point, more data is different.

Although I have taken my examples here from biology, the phenomena I have described extend well beyond that field into other domains of knowledge. Big Data has generated the most interest as a set of tools with huge commercial potential. As Dan Bouk argues in this volume, various forms of personal data have recently been imbued with added value and power.49 This value depends not only on an increased ability to aggregate data but also on new kinds of algorithms and ways of using them. And indeed, we can see versions of working knowledge present in some of the most commercially important algorithms. Google’s “Adsense” and “Adwords,” for instance, are algorithms that place targeted advertisements on websites.50 These algorithms strive to match the numerous businesses that wish to advertise online with the many websites that, for a fee, are available to host ads (e.g., online news sources, social media sites, blogs). The aim of Adsense/Adwords is to optimally connect advertisers to those who wish to display advertisements. Because Google is paid only when a Web surfer clicks on an ad, it has an interest in placing ads on sites where they are most likely to receive clicks. Usually, Google assumes, such clicks will occur where the advertisement is relevant (in some way or another) to the content of the Web page being viewed.

Posed in this way, Google faces a very big data problem. Given all the varied content of the web, how is it possible to decide whether a given website is relevant to a given topic? In other words, Google needed to find a way of determining the meaning of a website without actually having anyone read it (there are far too many websites for that to be practical). One of the most important parts of the system that Google uses to achieve this is called PHIL: Probabilistic Hierarchical Inferential Learner.51 It is possible to describe in broad and conceptual terms how PHIL (and Adsense/Adwords) works. However, such an understanding does not make it possible to predict which ads will be assigned to which websites. This is only practically knowable by letting the algorithm loose on the Web and seeing what it does. Like an ANN, even knowing the specific weightings that PHIL uses to link words into clusters and meta-clusters is

49 Dan Bouk, “The History and Political Economy of Personal Data over the Last Two Centuries in Three Acts,” in this volume.
not going to help in understanding why certain ads appear in certain places online. Indeed, Google’s business implicitly relies on this unpredictability—if it was possible to know how to attract particularly valuable ads, bloggers and websites would adjust their content in order to gain advertising revenue from their site. Instead, some advertisers and websites attempt to “game” Google’s algorithms by experimenting with them—by adjusting their content to attract more ads and more money. In practice, this is the only way it is possible to learn about the behavior of PHIL/Adsense/Adwords algorithms. Gaining a working knowledge of Google’s algorithms in this way can be a financially valuable undertaking.

In each case discussed here, algorithms and data structures provide ways of thinking about and solving biological problems that have now become part of the woodwork not only of biology but also of many parts of the Web and other information technologies we interact with every day. Interacting with algorithms and data structures allows us to gain knowledge of genomes or the Web or other large and complex data sets. The significance of Big Data lies not so much in its size or speed or scope, but rather in the particular ways in which it produces knowledge through what I have described here in terms of “working knowledge” or a “feeling for the algorithm.”

Arguing that technologies such as algorithms or data structures are important in shaping the kind of knowledge that emerges from them does not amount to technological determinism. Someone designed such algorithms and data structures. These coders and designers no doubt had specific goals and interests and styles of reasoning that made their software the way it is. This essay, as much as anything else, is an attempt to call attention to these (often hidden) processes of design and how they shape subsequent knowledge production. This shaping takes place through an interaction between human designers, coders, and users and their machine-based tools. If new biological knowledge is based on computers—if it somehow “belongs to” computers—then one important task is to begin to interrogate these computational structures in detail, including the histories of their production by humans. Who built them? For what purpose? Why were they built in one particular way and not another?

Understanding the history of Big Data requires insight into the histories of these algorithms and data structures—the history of machine learning, the history of algorithms like GigAssembler and Adsense, the history of hypertext. The history of Big Data is tied to the history of these algorithms and data structures. Big Data practices are not entirely novel—rather, they have a history that is rooted in the kinds of stories about algorithms and data structures that I have sketched here. These histories overlap with, but are importantly different from, histories of noncomputational, paper-based, “small data” practices. To understand the political, social, and cultural meanings of Big Data, we need to apply the kinds of sociological, historical, and political analysis that we have applied to labs and scientists to the insides of algorithms and data structures.

52 In fact, there is an industry based on these practices called “search engine optimization.”