LEARN LIKE A HUMAN

WHY CAN'T A COMPUTER BE MORE LIKE A BRAIN?

BY JEFF HAWKINS

By the age of five, a child can understand spoken language, distinguish a cat from a dog, and play a game of catch. These are three of the many things humans find easy that computers and robots currently cannot do. Despite decades of research, we computer scientists have not figured out how to do basic tasks of perception and robotics with a computer.

Our few successes at building "intelligent" machines are notable equally for what they can and cannot do. Computers, at long last, can play winning chess. But the program that can beat the world champion can't talk about chess, let alone learn backgammon. Today's programs—at best—solve specific problems. Where humans have broad and flexible capabilities, computers do not.

Perhaps we've been going about it in the wrong way. For 50 years, computer scientists have been trying to make computers intelligent while mostly ignoring the one thing that *is* intelligent: the human brain. Even so-called neural network programming techniques take as their starting point a highly simplistic view of how the brain operates.

In some ways, the task has been wrongly posed right from the start. In 1950, Alan Turing, the computer pioneer behind the British code-breaking effort in World War II, proposed to reframe the problem of defining artificial intelligence as a challenge that has since been dubbed the Turing Test. Put simply, it asked whether a computer, hidden from view, could conduct a conversation in such a way that it would be indistinguishable from a human.

So far, the answer has been a resounding no. Turing's behavioral framing of the problem has led researchers away from the most promising avenue of study: the human brain. It is clear to many people that the brain must work in ways that are very different from digital computers. To build intelligent machines, then, why not understand how the brain works, and then ask how we can replicate it?

My colleagues and I have been pursuing that approach for several years. We've focused on the brain's neocortex, and we have made significant progress in understanding how it works. We call our theory, for reasons that I will explain shortly, Hierarchical Temporal Memory, or HTM. We have created a software platform that allows anyone to build HTMs for experimentation and deployment. You don't program an HTM as you would a computer; rather you configure it with software tools, then train it by exposing it to sensory data. HTMs thus learn in much the same way that children do. HTM is a rich theoretical framework that would be impossible to describe fully in a short article such as this, so I will give only a high level overview of the theory and technology. Details of HTM are available at http://www.numenta.com.

ONE OF THE BAFFLING ASPECTS OF THE BRAIN IS THAT IT DECIDES WHAT TO LEARN ON ITS OWN

FIRST, I WILL DESCRIBE the basics of HTM theory, then I will give an introduction to the tools for building products based on it. It is my hope that some readers will be enticed to learn more and to join us in this work.

We have concentrated our research on the neocortex, because it is responsible for almost all high-level thought and perception, a role that explains its exceptionally large size in humans—about 60 percent of brain volume [see illustration "Goldenrod"]. The neocortex is a thin sheet of cells, folded to form the convolutions that have become a visual synonym for the brain itself. Although individual parts of the sheet handle problems as different as vision, hearing, language, music, and motor control, the neocortical sheet itself is remarkably uniform. Most parts look nearly identical at the macroscopic and microscopic level.

Because of the neocortex's uniform structure, neuroscientists have long suspected that all its parts work on a common algorithm—that is, that the brain hears, sees, understands language, and even plays chess with a single, flexible tool. Much experimental evidence supports the idea that the neocortex is such a general-purpose learning machine. What it learns and what it can do are determined by the size of the neocortical sheet, what senses the sheet is connected to, and what experiences it is trained on. HTM is a theory of the neocortical algorithm. If we are right, it represents a new way of solving computational problems that so far have eluded us.

Although the entire neocortex is fairly uniform, it is divided into dozens of areas that do different things. Some areas, for instance, are responsible for language, others for music, and still others for vision. They are connected by bundles of nerve fibers. If you make a map of the connections, you find that they trace a hierarchical design. The senses feed input directly to some regions, which feed information to other regions, which in turn send information to other regions. Information also flows down the hierarchy, but because the up and down pathways are distinct, the hierarchical arrangement remains clear and is well documented.

As a general rule, neurons at low levels of the hierarchy represent simple structure in the input, and neurons at higher levels represent more complex structure in the input. For example, input from the ears travels through a succession of regions, each representing progressively more complex aspects of sound. By the time the information reaches a language center, we find cells that respond to words and phrases independent of speaker or pitch.

Because the regions of the cortex nearest to the sensory input are relatively large, you can visualize the hierarchy as a tree's root system, in which sensory input enters at the wide bottom, and high-level thoughts occur at the trunk. There are many details I am omitting; what is important is that the hierarchy is an essential element of how the neocortex is structured and how it stores information.

HTMs are similarly built around a hierarchy of nodes. The hierarchy and how it works are the most important features of HTM theory. In an HTM, knowledge is distributed across many nodes up and down the hierarchy. Memory of what a dog looks like is not stored in one location. Low-level visual details such as fur, ears, and eyes are stored in low-level nodes, and high-level structure, such as head or torso, are stored in higher-level nodes. [See illustrations, "Everyone Knows You're a Dog" and "Higher & Higher."] In an HTM, you cannot always concretely locate such knowledge, but the general idea is correct.

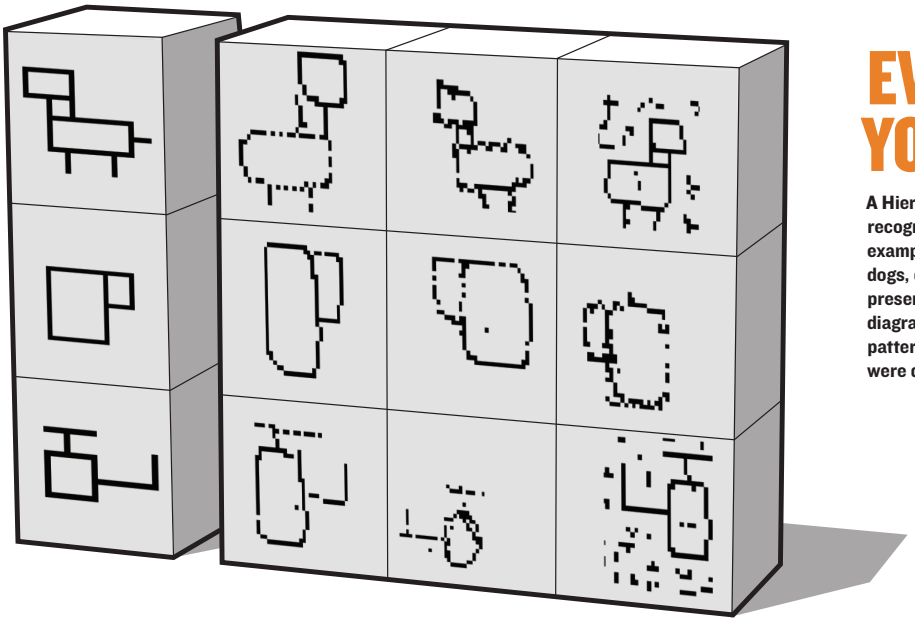
Hierarchical representations solve many problems that have plagued AI and neural networks. Often systems fail because they cannot handle large, complex problems. Either it takes too long to train a system or it takes too much memory. A hierarchy, on the other hand, allows us to "reuse" knowledge and thus make do with less training. As an HTM is trained, the low-level nodes learn first. Representations in high-level nodes then share what was previously learned in low-level nodes.

AN HTM IS DYNAMIC—IT DOESN'T DECIDE IN ADVANCE WHAT A NODE SHOULD LEARN

For example, a system may take a lot of time and memory to learn what dogs look like, but once it has done so, it will be able to learn what cats look like in a shorter time, using less memory. The reason is that cats and dogs share many low-level features, such as fur, paws, and tails, which do not have to be relearned each time you are confronted with a new animal.

The second essential resemblance between HTM and the neocortex lies in the way they use time to make sense of the fast-flowing river of data they receive from the outside world. On the most basic level, each node in the hierarchy learns common, sequential patterns, analogous to learning a melody. When a new sequence comes along, the node matches the input to previously learned patterns, analogous to recognizing a melody. Then the node outputs a constant pattern representing the best matched sequences, analogous to naming a melody. Given that the output of nodes at one level becomes input to

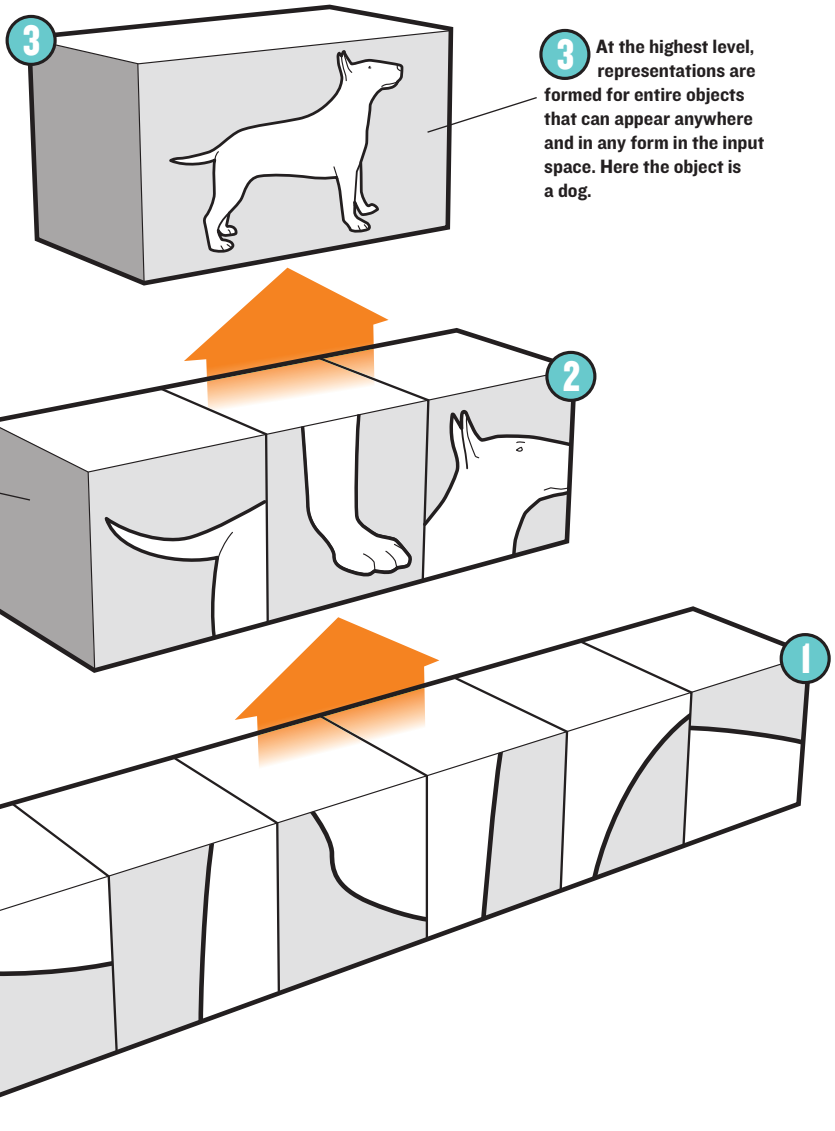
PREVIOUS PAGE: BRYAN CHRISTIE DESIGN



EVERYONE KNOWS YOU'RE A DOG

A Hierarchical Temporal Memory (HTM) is trained to recognize patterns by being exposed to a large number of examples of a particular kind of object—in this illustration, dogs, cups, and helicopters. (The input patterns must be presented as moving in time, something not shown in the diagram.) After training, the HTM correctly classified patterns to which it had not been exposed even when they were distorted, translated, or marred by noise.

HIGHER & HIGHER



3 At the highest level, representations are formed for entire objects that can appear anywhere and in any form in the input space. Here the object is a dog.

2 As information flows up to higher levels in the hierarchy, the activity represents increasingly complex structures comprising ever-larger areas of the input. The second layer depicts a tail, a paw, and a head.

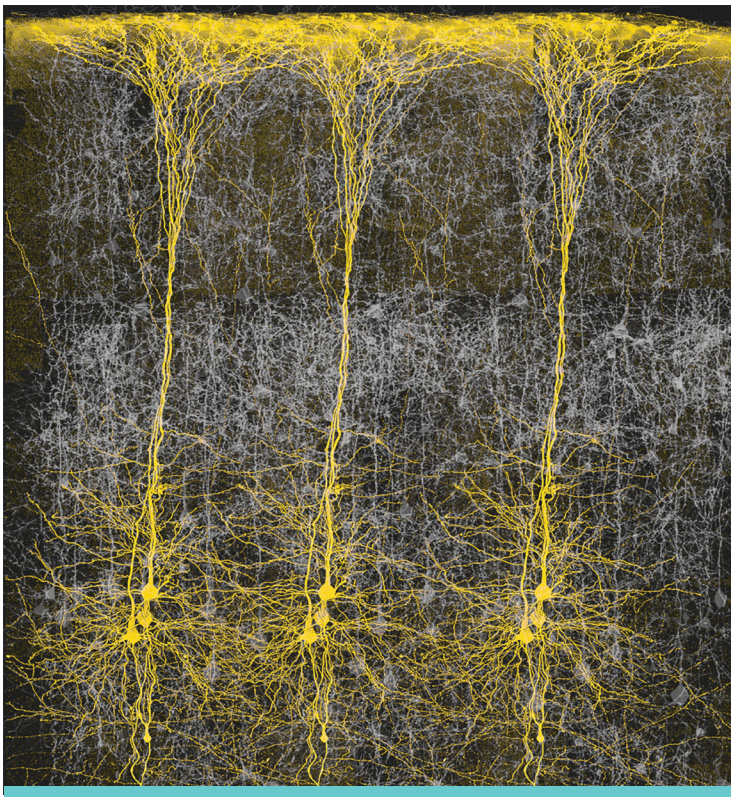
1 The "H" in HTM stands for hierarchical, meaning that only the nodes in the lowest level of the hierarchy are connected directly to external data, such as pixels from a camera or temperature readings from sensors in a weather-modeling system. The bottom layer of this illustration represents the input data as simple contours.

JASON LEE

nodes at the next level, the hierarchy learns sequences of sequences of sequences.

That is how HTMs turn rapidly changing sensory patterns at the bottom of the hierarchy into relatively stable thoughts and concepts at the top of it. Information can flow down the hierarchy, unfolding sequences of sequences. For example, when you give a speech, you start with a sequence of high-level concepts, each of which unfolds into a sequence of sentences, each of which unfolds into a sequence of words and then phonemes.

Another, subtler way an HTM exploits time is how it decides what to learn. All of its parts learn on their own, without a programmer or anyone else telling the neurons what to do. It is tempting for us to try to fill such a coordinating role by deciding in advance what a node should do, for instance by saying, "Node A will learn to recognize eyes and ears, and node B will learn noses and fur." However, that approach does not work. As nodes learn, they change their output—which affects the input to other nodes. Because memory in an HTM is dynamic, it is not possible to decide in advance what a node should learn.



GOLDENROD: The neocortex, the seat of human reasoning, is an astonishing organ. Its 30 billion or so neurons are organized in six layers, each about the thickness of a playing card. The picture here, an image from the Blue Brain Project, shows neurons from the fifth layer. The Blue Brain Project is a joint research effort by IBM and the Ecole Polytechnique Fédérale de Lausanne to study the brain using IBM's Blue Gene supercomputing system.

So how does a node know what to learn? This is where time plays a critical role and is one of the unique aspects of HTM. Patterns that occur close together in time generally have a common cause. For instance, when we hear a sequence of notes over and over, we learn to recognize them as a single thing, a melody. We do the same with visual and tactile patterns. Seeing a dog moving in front of us, for example, is what teaches us that a left-facing dog is actually the same as a right-facing dog, in spite of the fact that the actual information on the retina is different from moment to moment. HTM nodes learn similarly; they use time as a teacher. In fact, the only way to train an HTM is with input that changes over time. How that is done is the most challenging part of HTM theory and practice.

Because HTMs, like humans, can recognize spatial patterns such as a static picture, you might think that time is not essential. Not so. Strange though it may seem, we cannot learn to recognize pictures without first training on moving images. You can see why in your own behavior. When you are confronted with a new and confusing object, you pick it up and move it about in front of your eyes. You look at it from different directions and top and bottom. As the object moves and the patterns on your retina change, your brain assumes that the unknown object is not changing. Nodes in an HTM assemble differing input patterns together under the assumption that two patterns that repeatedly occur close in time are likely to share a common cause. Time is the teacher.

The final word in HTM is "memory." This attribute distinguishes HTMs from systems that are programmed. Most of the effort in building an HTM-based system is spent in training the system by exposing it to sensory data, not in writing code or configuring the network. Some people assume memory means a single remembered instance, such as "what I ate for lunch." Others associate memory with computer memory. In the case of HTM, it is neither. HTMs are hierarchical, dynamic, memory systems.

WHAT MAKES HTM different from other approaches to machine learning? HTMs are unique not because we have discovered some new and miraculous concept. HTM combines the best of several existing techniques, with a few twists thrown in. For example, hierarchical representations exist in a technique called Hierarchical Hidden Markov Models. However, the hierarchies used in HHMMs are simpler than those in HTM. Even though HHMMs can learn complex temporal patterns, they do not handle spatial variation well. It is as if you could learn melodies but not be able to recognize them when played in a different key. Still, the similarity between HTM and other approaches is a good sign: it means that other people have reached similar conclusions. A detailed comparison to other techniques is available on Numenta's Web site.

Another unique aspect of HTM is that it is a biological model as well as a mathematical model. The mapping between HTM and the detailed anatomy of the neocortex is deep. As far as we know, no other model comes close to HTM's level of biological accuracy. The mapping is so good that we still look to neuroanatomy and physiology for direction whenever we encounter a theoretical or technical problem.

Finally, HTMs work. "If we really understand a system we will be able to build it," said Carver Mead, the famous Caltech electrical engineer. "Conversely, we can be sure that we do not fully understand the system until we have synthesized and demonstrated a working model." We have built and tested enough HTMs of sufficient complexity to know that they work. They work on at least some difficult and useful problems, such as handling distortion and variances in visual images. Thus we can identify dogs as such, in simple images, whether they face right or left, are big or small, are seen from the front or

IBM/EPFL

the rear, and even in grainy or partially occluded images.

We have been pleased with the results so far and have confidence that HTMs can be applied to many other problems we have not tried. We also know of problems that our current implementation of HTM will not solve today, and we have yet to build very large networks. Time will tell what obstacles we will encounter in the future, but for now we are encouraged by how well the technology is working.

THE IDEA OF MARRYING computer science with brain research came to me soon after I graduated in electrical engineering from Cornell University, in 1979. For many years I pursued the interest part time while working as an engineer for a number of companies (including two I cofounded—Palm Computing and Handspring), and for one year I worked at it full time, as a graduate student in biophysics at the University of California at Berkeley.

In 2002, with the encouragement of some neuroscientist friends, I created the Redwood Neuroscience Institute. For three years, I worked with about 10 other scientists there on all aspects of neocortical anatomy, physiology, and theory. During those years, more than 100 other scientists visited the RNI for discussion and debate, and I made progress on HTM theory.

AS CARVER MEAD ONCE SAID, "IF WE REALLY UNDERSTAND A SYSTEM, WE WILL BE ABLE TO BUILD IT"

By 2004 I had developed the essence of HTM, but the theory was still rooted in biology (I published that biological theory in 2004 in my book *On Intelligence*). I did not know how to turn the biological theory into a practical technology. A colleague of mine, Dileep George, was aware of my work and created the missing link. He showed how HTM could be modeled as a type of Bayesian network, a well-known technique for resolving ambiguity by assigning relative probabilities in problems with many conflicting variables. George also demonstrated that we could build machines based on HTM.

His prototype application was a vision system that recognized line drawings of 50 different objects, independent of size, position, distortion, and noise. Although it wasn't designed to solve a practical problem, it was impressive, for it did what no other vision system we were aware of could do.

In 2005, with a theory of the neocortex, a mathematical expression of that theory, and a working prototype, George and I decided to start Numenta, in Menlo Park, Calif. Our experience in industry and academia taught us that people move more quickly in industry, especially if there is an opportunity to build exciting products and new businesses. Today, the RNI continues as the Redwood Center for Theoretical Neuroscience at UC Berkeley. George and 15 other employees work at Numenta, and I split my time between Numenta and Palm.

TO HELP NUMENTA jump-start an industry built on the ideas of HTM, we embarked on creating a set of tools that allow anyone to experiment with HTMs and to use HTMs to solve real-world problems. By making the tools broadly available, providing source code to many parts of the tools, and encouraging others to extend and commercialize their applications and enhancements, we hope to attract engineers, scientists, and entrepreneurs to learn about HTM. The tools together constitute an experimental platform for HTM that is meant to attract many developers by giving them the chance to create successful businesses.

We realize it will take a lot of time and effort to learn an entirely new way of computing. One of the things I have learned is to never become an obstacle to a developer working hard on a new platform. To this end we have created lots of documentation and several examples. We provide complete source code for two of the platform's three components.

The first part is the run-time engine. This component is a set of C++ routines that manage the creation, training, and running of HTM networks on standard computer hardware. It's highly scalable, meaning you can run the entire platform on anything from a laptop with a single CPU to PCs with multiple cores, to a large computer cluster. Significant development can be done on a single CPU; however, as applications grow in size, multiple CPUs may be needed for performance.

Today the run-time engine runs on Linux. Our employees and customers use both PCs and Macs. When designing HTM-based systems, the developer needs to experiment with different configurations. Running concurrent tests saves a lot of time. Our tools make parallel testing easier.

The run-time engine guarantees that a developer need not worry about such things as how messages are passed between nodes or the sharing of data between machines. When the run-time engine is running an HTM network on a cluster of servers, it automatically and efficiently handles the message processing back and forth. By taking care of the plumbing, the engine frees the developer to focus on the HTM design, training, and results.

WE HAVE BUILT ENOUGH HTMS TO KNOW THAT THEY WORK

The second platform component is a set of tools HTM developers use to create, train, and test a network. The tools are written in the Python scripting language, the source code for which is available for inspection and modification, along with documentation of the application programming interfaces, or APIs, to the run-time engine.

Although the first set of tools is sufficient, some people probably will want to modify and enhance them. They might, for example, want one tool to visualize the data structure of an HTM node of a vision system and a slightly different one for an HTM that analyzes business data. We also can imagine value-added resellers packaging sets of tools and pretrained HTMs for particular applications or for particular sensor systems, such as radar or an infrared camera.

Developers may package and sell tools derived from Numenta's source code free of charge. Our one requirement is that the tools be used only with Numenta's run-time engine.

The third part of the platform is a plug-in API and associated source code, which together enable a developer to create entirely new kinds of nodes and plug them into a network. Nodes are of two types: a basic learning node (which might appear anywhere in the network) and an interface node (which leads out of the network to sensors that provide input or to effectors that accept output). Numenta provides a basic learning node and several sensor nodes. When you create a network,

you select one of the types for each node in the network, in some cases modifying them so that they can connect to a new type of sensor.

People with sufficient math skills might want to improve the learning algorithms. We expect that such enhancements will go on for years. Numenta is making available the source code to its existing nodes so that developers can understand how they work and improve them if they want. Developers can freely sell nodes derived from Numenta's source code without paying anything to Numenta. Our only requirement, again, is that nodes derived from Numenta's source code be used only with Numenta's run-time engine.

Finally, we have created a set of sample HTM networks, as well as documentation and training materials to help developers get started.

WHAT CAN YOU DO with Numenta's platform? Through many discussions with developers, we have determined that HTMs can potentially be applied to a wide range of applications. For example, there is a class of problems similar to recognizing the contents of a picture—which we call spatial inference. Given a novel image or pattern, the HTM tells you what it is. The data may come from a camera or other type of sensor such as laser radar. Car manufacturers have several such applications.

We have also seen many applications that model networks, including computer networks, power networks, networks of machines, and even social networks. In such applications, the HTM learns to recognize or predict undesirable future conditions in a given network.

HTMs have their work cut out for them in any endeavor to make sense of data sets. Oil exploration furnishes an example, because large amounts of data must be collected and analyzed to decide where best to drill a well. Pharmaceutical firms are interested in knowing whether HTMs can help them discover new drugs by teasing out clues from the data they have amassed in drug trials.

HTMs work best when there is hierarchical structure in the data. Businesses have such structure—layers of managers, supervisors, and workers are hierarchically organized, as are business functions such as finance, sales, and accounts receivables. We were approached by a large business-consulting firm that wanted to use our tools to study corporate behavior. Can we model business functions with an HTM? We don't know yet, but I think so, even though it's not one of the classic AI problems.

With many applications of HTM, the trickiest part may be in deciding what are the "sensory" data to train on and how to present them in a time-varying form.

We also have seen interest in modeling manufacturing processes. Many industries have lots of data without an easy way to discover the patterns in the data or a means to make predictions based on past experience.

Those and other potential applications for HTMs are described in documents on Numenta's Web site and in my book.

There are applications we know we cannot solve today, given the state of the platform. Anything involving long memory sequences or specific timing cannot be supported by our current node algorithms. For example, understanding spoken language, music, and robotics requires precise timing. This constraint is not a fundamental limitation of the Numenta platform but rather a limit of the tools and algorithms we have at present. We plan to enhance the algorithms to add those capabilities in the future.

THIS VERSION OF NUMENTA'S platform is called a research release, in contrast to the usual alpha or beta release, although the standards of quality and documentation would suffice to make it a solid beta. We have chosen the term "research release" because we find most people take months to master the concepts of HTM and become fully productive with the platform.

By analogy, say you were asked to write a complex software program such as a spreadsheet application. Further, suppose that you never had been exposed to computers, computer concepts, compilers, or debuggers. It would take you months of experimentation and learning before you could properly start work on the spreadsheet program. The concepts behind HTM and the tools we have created are similarly somewhat foreign.

In addition, our first learning algorithms are not as easy to use as we would like. They have a few parameters whose appropriate values need to be determined by experimentation and analysis. That isn't too hard, but it isn't trivial either. We believe we can eliminate this step in future releases, but for now it is required for most applications. Although it is simple to install the platform, easy to get started, and often fun, a user of Numenta's first release should expect to spend several months to a year experimenting before getting commercially useful results.

Although we will likely charge for our tools in the future, the research release is available at no charge.

HTM is not a model of a full brain or even the entire neocortex. Our system doesn't have desires, motives, or intentions of any kind. Indeed, we do not even want to make machines that are humanlike. Rather, we want to exploit a mechanism that we believe to underlie much of human thought and perception. This operating principle can be applied to many problems of pattern recognition, pattern discovery, prediction and, ultimately, robotics. But striving to build machines that pass the Turing Test is not our mission.

The best analogy I can make is to go back to the beginning of the computer age. The first digital computers operated on the same basic principles as today's computers. However, 60 years ago we were just starting to understand how to use computers, what applications were best matched to them, and what engineering problems had to be solved to make them easier to use, more capable, and faster. We had not yet invented the integrated circuit, operating systems, computer languages, or disk drives. Our knowledge of HTMs today is at a similar stage in development.

We have recognized a fundamental concept of how the neocortex uses hierarchy and time to create a model of the world and to perceive novel patterns as part of that model. If we are right, the true age of intelligent machines may just be getting started. ■

ABOUT THE AUTHOR

JEFF HAWKINS, inventor of the Palm Pilot, is the founder of Palm Computing, Handspring, and the Redwood Neuroscience Institute. In 2003 he was elected a member of the National Academy of Engineering.

TO PROBE FURTHER

Hawkins's book *On Intelligence*, cowritten with Sandra Blakeslee, was published by Times Books in 2004. The book's companion Web site is at http://www.onintelligence.org.

Learn more about Numenta at http://www.numenta.com. The Redwood Center for Theoretical Neuroscience is at http://redwood.berkeley.edu.