American Society for Investigative Pathology – ASIP.org

MODALITY OF THE MONTH
A MONTHLY METHODOLOGY REVIEW FROM THE ASIP
DIGITAL AND COMPUTATIONAL PATHOLOGY SCIENTIFIC INTEREST GROUP

November 29, 2016

MoM#6: COMPUTATION AND MACHINE LEARNING
Stanley Cohen, MD

Introduction

All of the MoMs covered to date involve extensive computation; for example, in MRI, data about proton spin are processed to yield highly detailed 3D images. More generally, computation allows us to work with large data sets and extract meaningful patterns, relationships, and meanings from them. Computation has become an important part, not only of pathology, but of all medicine and science. The ultimate culmination of this trend is to enable the computer to suggest specific diagnostic and therapeutic possibilities for human review and action. Many names have been attached to this: machine learning, deep learning, hierarchical learning, and artificial intelligence, for example, but there is no consensus as to whether these are distinct aspects of machine "intelligence." This is not surprising, as there is no agreed upon formal definition of human intelligence. In this MoM we will try to provide a conceptual overview of work in this area. While this is not meant to be a primer on computers or computer programming, it is important to understand a few basics.

Basics

When stripped of the mystique and jargon, there are three basic parts of a computer. Core memory where programs and data are stored, a central processing unit that executes the instructions and returns the output of that computation to memory for further use, and devices for input and output of data. Similarly, there are three basic concepts that are intrinsic to every programing language: assignment, conditionals, and loops.

Assignment: The notion that gives most computer newbies pause is "assignment." The combined statements (G = 5) and (G = G+2) are mathematically inconsistent but make perfect sense in computer programming. In computer language, G is a location into which the number 5 is placed, and not the number 5 itself. These are statements and not equations. The first statement (G = 5) means that 5 is stored in location G. The next statement (G = G + 2) means that we add 2 to whatever is in G (in this case 5), and the new value (now 7) replaces the old value in G. It helps to think of the = sign as meaning that the quantity on the left is replaced by the quantity on the right.

Conditionals: In contrast to assignment, conditionals are easy to understand, which is fortunate, since the conditional or IF command is probably the most important instruction of every programming language. It works like this. We ask the computer to make a test that has two possible results. If one result holds then the computer will do something. If not, it will do something else. For example, "IF you are wearing a purple neon T-shirt, print "you have terrible taste". IF NOT, print "there is hope for your fashion sense." This is known as an IF statement. In more general terms, IF a condition holds, do something. If not, do something else. The somethings can be any operation or series of operations or move the computer to a different part of the program. It is the basis of a class of machine learning called the Random Forest.

Loop: The third concept is that of a LOOP. This enables the computer to perform a single set or set of steps a given number of times. For example, FOR X = 1 to 100, print sqrt(x) NEXT X will print the square roots of the numbers 1 through 100.

The specific words used to invoke conditionals or loops are dependent on the computer language that is being used for programming. A working knowledge of at least one computer language would be helpful for anyone who wishes to explore this field more fully. Python is a good start because it is easy to learn and shields the user from some of the more technical details incorporated into other languages (1,2). Python, though simple, is powerful; with it one can construct simple neural networks with minimal effort in order to better understand some of the principles underlying machine learning (3).

There are many different modes of machine learning, including rules models, linear models, probabilistic models, logical models, geometric models, and distance models. There can both supervised and unsupervised learning in each. A typical definition of machine learning that encompasses these is that it is based on a set of algorithms that attempt to model high level abstractions in data by using multiple layers of analysis. Much of this is beyond the scope of this article, but can be found in (4). In this discussion, we will focus on a small number of basic machine learning models to understand their underlying structures and strategies.

Rules Models

One of the earliest forms of machine learning involved Rules Models. This is like giving instructions or rules to a baby or toddler: “spinach is good for you,” “always do what Mommy and Daddy say,” “only cross streets when the light is green.” The baby also needs to know the meaning of words. With enough vocabulary and rules, a complete pattern of cognitive thought emerges that we call intelligent behavior (except for adolescence). A similar approach can be taken with computers. To program a computer to play chess, we can give it the rules of the game, some basic concepts like “it’s good to occupy the center” and data as to classic responses to typical openings. We can also define rules for evaluating every position on the board at every move. The machine can then calculate all possible moves from a given position and select those that lead to some advantage (pieces captured, positional superiority, etc.). It can also calculate the various responses to each such move by its opponent. From the set of near optimal moves it can then select those moves that not only appear good for it, but also minimize the opponent’s opportunities to improve its position in the next move. This means calculating billions of possible moves. A computer does this by brute force. Decades ago, Deep Blue beat Gary Kasparov using a rules based model by searching several million positions per second. Kasparov was probably limited to about 5 per second. Rule based systems use rules prepared by humans.

A good chess player does not have to calculate every move, but rather has a sense of the best lines of play. Also, a good player can “see” several moves ahead by him/herself and the opponent. He/she can do so by a sophisticated sense of “good” and “bad” lines of play based, in part, upon the experience of learning from many other games played previously. In 2015, an artificial intelligence machine called Giraffe taught itself to play chess by evaluating positions much more like humans using a neural network model. After about 72 hours of play it matched the best rules based chess engines in the world, as it shaped its internal parameters based on training examples. Networks such as these use sophisticated algorithms to make judgments without explicit input as to the rules to be invoked in making those judgments. We will now discuss some of these in detail.

Decision Trees and Random Forests

In essence, a machine learning program in these models takes a set of data that is capable of being separated into two or more categories, based upon a set of if-then decisions (such as those described above). A data set that is capable of such categorization is known as linear separable. An example is a data set of fruit consisting of apples and bananas. A simple way of a having program separate these is the following: IF the fruit is yellow, call it a banana. IF NOT, call it an apple. Unfortunately, some apples are also yellow, so a second test is necessary. If the fruit is round, call it an apple. If NOT call it a banana. In this way, we can generate a decision tree with multiple branch points. In this case, shape and color. Unfortunately for our simple model, the definition of "roundness" is a bit fuzzy. We will also need a way of characterizing roundness. If this involves multiple parameters, then the decision tree expands. Also, since the definition of roundness is fuzzy, we may only be able to estimate a likelihood of roundness for each of those parameters. Thus, each branch of a decision point may have a probability value rather than an absolute yes or no decision. We can "train" such a simple model by exposing it to a test sample (a subset of the original data set), and compare its results to the correct answer. The program can be written so that the machine can adjust its probabilities, or "weightings," to maximize its correct choices. This is one of the the simplest forms of machine learning.

A more elegant model is the Random Forest Model. Suppose we have a set consisting of a large number of different kinds of cars and we want to separate them by specific type, so that, for example, we can correctly identify all the red Porsche Carreras. For the sake of this simple example, assume that designating labels (such as the brand name or model name) are not present. There are a number of other features we could use: size, shape, number of doors, horsepower range, price, etc. Useful real-life models are usually even more complex. Our model will vary with the number of if-then decisions, the likelihood of satisfying each of them, and in some case the sequence of the if-then decisions. Thus, even though the model is theoretically trainable, we can never be sure that we have the best possible model. Suppose, however, we created a decision tree based on a subset of the features rather than all of them. Now suppose we created a second decision tree using a different subset of features, and so on. We can also generate our data by using a different subset of the data for each decision tree. Clearly, there are two ways we can insert randomness into a Forest based (i) on what data are selected for each tree, and (ii) how the criteria for the splits are chosen. Usually, a Random Forest uses the square root of the number of features as the maximum features that it will look for on any decision branch. The way in which the program chooses the best of the various parameters we discussed is outside the scope of this discussion, but ultimately, we can average, in some manner, the results of all these trees, which we then refer to collectively as a Random Forest (because some of the choices are based on probabilistic values). A good overview of trees and random forests can be found in (5).

Support Vector Machines

We can think of a dividing boundary between two subclasses of objects as a decision surface. Support vectors are those data points that lie closest to the decision surface. Because they are close they are the most difficult to classify. In two dimensions, with a cluster of red dots on the left and a cluster of blue dots on the right, it may be possible to define more than one cut-off line that totally separates them. For higher dimensions, with multiple criteria, we would need a hyperplane rather than a line. However, as we get samples closer and closer to this boundary, it may turn out that one of the cut-offs may be more efficient at discriminating these classes than the other. By definition, support vectors are the elements of the training set that would change the position of the dividing hyperplane if removed, and thus, support vectors are the critical elements of the training set. The problem of finding the optimal hyperplane requires a sophisticated

mathematical and theoretical underpinning and is beyond the scope of this review. One of the more readily accessible reviews is (5).

Neural Networks (learning rules by creating hidden variables from data)

Random Forests are not very good for problems in which criteria are not easy to define in either a definitive or probabilistic way. A good example is handwriting recognition. It is relatively easy to train a Random Forest to correctly identify letters that are printed in a book, but extremely difficult, if not impossible to do so when a human being is writing those letters, because of the huge variation in handwriting patterns and styles. The brain is very good at this because it makes use of a series of processing levels, each one doing more complex analysis than the previous one. Each of these levels is composed of many interconnected neurons with the firing patterns of each determined by that of its neighbors. A Neural Network is a model composed of artificial neurons (either physically implemented or simulated in code) that function in a similar manner. These artificial neurons fall into two categories, the perceptron and the sigmoid neuron. A perceptron is an element that takes several binary (0 or 1) inputs (X1, X2, X3….Xn) and produces a single binary output (0 or 1) depending upon the sum of the inputs. It is important to note that all the the inputs need not be weighted equally to compute the output. For each input a "weight" is required, namely a number expressing the importance of the respective input to the output. The weight can range anywhere between 0 and 1 or can be scaled over another interval. For each perceptron, there is a threshold that must be exceeded by the sum of inputs in order to the neuron to actually emit an output, or "fire". This threshold is defined as its "bias," and is dependent on the weighting scale.

The power of a neural network is that the program itself can adjust these parameters (such as weights and thresholds) as it improves its results on the basis of increasingly accurate results on a training subset of the data. Ideally, it would be nice if it could reach the point where a small change in these variables caused only a small change in accuracy, so that the system can be trained in finite time with a manageable set of data. The problem is that a small change in a single perceptron may cause the output to completely flip, for example from a zero to a one, which could then completely change the behavior of the network in some complicated way. To avoid this, a new type of neuron called a sigmoid neuron is created. This is like a perceptron but modified so that small changes in both their weights and thresholds (bias) cause only a small change in output. Neither the inputs or outputs of a sigmoid neuron need be binary. They can take on any value between zero and one.

A neural network can compute whether you are likely to go to the next ASIP meeting! In a simple model, X1 can be the scientific value of going to the meeting, X2 can be the cost and convenience of transportation to get there, and X3 the likely weather. The weights represent the relative importance you assign to these parameters. You can have several perceptrons, with varying and possible overlapping inputs. These form a layer of the neural network. This simple example doesn't take into account many other parameters, nor the opportunity cost (namely what you could have been doing instead of going to the meeting). For a complex data set such as this, a single layer may not be able to analyze at the level of abstraction to provide a meaningful solution outcome for us. However, we can create a second layer of perceptrons that weighs up the results from the first layer of decision making. The outputs of the first layer become inputs for the second layer. Thus the perceptron can make a decision at a more complex and abstract level than the perceptrons in the first layer. Each new layer that is added creates the possibility for more complex decision making, not only in terms of numbers of variables but also in identifying critical relationships among these. In this way, we can generate complex neural networks of great depth to deal with the kinds of complex data sets and decision making that emerge from clinical and experimental studies. Thus, a typical neural network is organized into layered regions that function in a hierarchy of abstractions and each layer is

composed of elements that behave in a manner similar to biologic neurons. There is where the analogy to a brain arises. Further, as we indicated, the weightings for each input-output element are adjusted by the program until a desired level of recognition accuracy is obtained. Because of the huge number of interconnecting pathways, it is not usually possible to determine the exact parameters that achieve this final state. As in the case of a Random Forest, a neural network uses training data sets, but its internal processing of the data is, in a very real sense, hidden from the observer. This is also analogous to a biological brain.

In neural networks, the decision rules can be less restrictive than in a branching tree model in that we can avoid predefining criteria. In principle, one could simply show the network an apple and say this is an apple. Ditto for bananas. After a large enough training set, the network will adjust itself so that it can distinguish apples and bananas with high accuracy, even without the programmer explicitly defining such properties as roundness (6). Modern Neural networks allow for efficient learning of non-linear data, i.e., data that can not easily be stratified into decision tree-based models with vector support modules. However, support vector models can also be used with neural networks.

What we have described so far is the use of perceptrons as methods of weighing evidence. By defining inputs properly, we can also use the perceptron to compute logical functions such as AND, OR, and NAND. The NAND gate is a logic gate that produces an output that is false if all the inputs are true. The NAND gates are universal for logical computation since networks of NAND gates can perform any logical operation no matter how complicated. Moreover, since computation itself is based on logic gates with binary values, a neural network could, in theory, be used to facilitate the design of more complex neural networks.

Summary and Overview

Advances in Artificial Intelligences are dependent upon the continuing evolution of more powerful computers and by the increasing interest in this field by new generations of computer programmers (the artificial intelligentsia). Modern techniques such as genetic algorithmic learning will also play a role. In this approach, one can take a simple programmed solution based upon a set of algorithms and make copies of the starting point, randomly altering components of the algorithm. We can then select those altered versions that do the best at solving the problem and use these as our new starting point and so on. Also, Insights about the structural and functional anatomy of the brain via optogenetics and other new techniques will illuminate (pun intended) these approaches. As efforts such as the Blue Brain Project evolve (about 31,000 neurons and 40 million synapses have already been copied)) we may be able to reverse-engineer our brains for greatly enhanced neural network computation. Regardless of where this all leads, it is clear that modern computer analysis goes beyond simple data mining. The only caveat is that where there is artificial intelligence there is the possibility of artificial stupidity. Thus standardization and validation must accompany all stages of implementation. When done properly, computer-assisted pathology of the future will not only be able to provide quantitative image analysis and correlate this with other large data-driven sets of clinical data, but also to make explicit “informed” suggestions regarding diagnosis, treatment, and outcomes prediction. Several decades ago, I gave a talk on the future of pathology in which I said that “the eye will always be one of the most important tools in our diagnostic armamentarium, but it is not certain that the eye will be a human one.“ I was only partially right. The human eye will always be important, but judgments based on what is seen by that eye will be vastly augmented by the computer. The computer will not replace the pathologist, but perhaps will become a valued and trusted pathologist’s assistant (or lab technician in a research setting). Artificial intelligence, at least for the foreseeable future, does not imply either consciousness of awareness (whatever those terms mean). It is possible to program a computer to make inferences and exhibit “volition” but that is not true

awareness, which I suspect is a basic requirement for positive and negative forms of creativity (although in the present political climate I sometimes have my doubts). There are those who believe that quantum computing (to be discussed in a future MoM) holds the promise of developing a conscious artificial intelligence because of the possibility of multiple complex computations performed simultaneously without the constraint of binary yes-no decision making and the introduction of quantum indeterminacy in the system. However, it will be exceedingly difficult to determine whether or not such a construct is conscious or aware until we have a better understanding of the nature of awareness itself. This dilemma is posed and (almost) convincingly answered in a recent film (7). A good analysis of the potentials and challenges of machine learning carried to this conclusion can be found in (8), but perhaps the best test of true artificial intelligence will occur when it achieves tenure.

FURTHER READING

1. Horstmann, C. and Necaise R. Python for Everyone, J. Wiley and Sons, Inc., 2016.
2. Sande, W. and Sande, C. Hello World: Computer Programming for Kids and Other Beginners, Manning, 2009.
3. Rashid, T. Make Your Own Neural Network", CreateSpace Publishers, 2016.
4. Plach, P. Machine Learning, Cambridge University Press, 2015.
5. Hartshorn, S. Machine Learning with Random Forests and Decision Trees, Kindle, 2016.
6. Berwick, R. An Idiot's Guide to Support Vector Machines". http://www.cs.ucf.edu/courses/cap6412/fall2009/papers/Berwick2003.pdf
7. http://www.rogerebert.com/reviews/ex-machina-2015
8. Lerner, E.M. A Mind of Its Own. Analog, September 2016, pp.38-49.