Regarding theory of the learnable and cortical computation

Manmeet Singh

University of California, Santa Cruz

March 12, 2016

Abstract

Humans are able to learn concepts through experience in a way that is completely unrelated to their genetic makeup (which largely only directs morphological changes). To study learning as acquision of new knowledge from a computational standpoint, Leslie Valiant puts forth a theory of the learnable. It consists of 3 parts: choosing an information gathering mechanism, the learning protocol, and exploring the class of concepts that can be learned in a reasonable (polynomial) number of steps. This theory is further discussed in the light of experimental neuroscience. A model of disjoint learning based on new information is considered which tends to support the general theory of learning put forth some time earlier. More generally, two useful regimes for neural computation are identified along with their quantative parameters.

1 Introduction

Part of what continues to fascinate us is our ability to learn to perform tasks that biology is virtually unrelated to, except providing the underlying "hardware". It is important to differentiate this type of learning from one's genetic component in that there is no explicit programming for it in any conventional sense. This commonplace phenomenon of learning deserves attention similar to the computability theory which became possible once mechanisms for modeling mechanical computation became possible. The problem here is to discover models which are interesting to study and hold promise to be relevant in both explaining human experience and building devices that can learn. Mainly it is shown to be possible to design *learning machines* that have three of the following properties:

1. Machines can learn classes of concepts. These classes can be distinguished individually.
2. The classes of concepts are appropriate and important for general-purpose knowledge.
3. The computational process to deduce the desired results requires a feasible (ie. polynomial) number of steps.

Excluding the third point, it can be seen that a learning machine consists of a *learning protocol* and *deduction procedure*. As the first source of information, the datasets chosen include a supply of typical data that positively exemplifies the concept. These examples are assumed to have a probabilistic distribution determined arbitrarily which re-enforces the concept each time upon call

of the routine EXAMPLES. The second source of knowledge is the routine ORACLE which tells the learner whether or not the given data positively exemplify the concept on each call. General knowledge is represented in terms of Boolean functions or a set of propositional variables. The recognition algorithms under discussion will therefore use Boolean circuits or expressions.

The conclusions derived from the theory of the learnable are that there are specific classes of concepts that are learnable in the polynomial time using the learning protocols described. These classes can be characterized by defining the class of programs that recognize them. In each of the classes, there are special kinds of Boolean expressions. These three classes are (1) conjunctive normal form expressions with bounded literals in each clause, (2) monotone disjunctive normal form expressions, and (3) arbitrary expressions with each variable occurring only once.

The three classes mentioned are discussed in the light of the deduction procedure mentioned. The deduction procedure outputs an expression that closely approximates the expression to be learned. The learning expression never says yes when it should say no, but can sometimes say no when it should say yes. This margin of error can be reduced substantially by increasing the probability space of positive examples. At a more broad level, it is shown that with this probabilistic notion of learning, highly convergent learning is possible for a whole classes of Boolean functions.

The theory of learnable is distinguishable from the diverse body of previous work in that the results presented highlight the difference between learning by being programmed (non-explicitly) and learning by being programmed.

In the latter part of the paper, we explore the application of computer science methodology to provide insight into the basic tasks of cognition. This consists of three parts: a concrete model of neural computation, explicit specifications of cognitive tasks to be realized, and explicit algorithms that can be executed on this model to realize the tasks. Additionally, since the brain can perform significant tasks in 100-200 milliseconds, basic algorithms need to work in very few steps.

The model of computation used here is defined by three parameters *n, d, k* where n is the number of neurons in the system, d is another system of some number of input neurons into n and k is the value for the excitatory postsynaptic potential (EPSP).

2 Learning Protocol and Learnability

A set of Boolean variables $p_1, .., p_t$ is considered. Additionally, there are no assumptions about the independence of these variables from one another. A vector is a set of t variables where each one has been assigned a value of 1, 0 or * where * denotes an *undetermined* value. A vector is considered total if each of the values within the set have been determined.

A function F is a mapping from the set of 2^t total vectors to a true or false value. The function F becomes a *concept* F if its domain (value of 1 or 0) is extended to set of all vectors. In simpler terms, a concept is a vector in which each Boolean expression has been determined and agrees with any other vector that contains a subset of the shared variables.

An arbitrary probability distribution D over the set of all vectors that map to True or 1 is considered. In other words, for each v where $F(v) = 1$, $D(v) \geq 0$ and also that $\sum D(v) = 1$. In general terms, D describes the relative frequency with which positive examples of F occur.

Based on the fundamental units of the learning protocol described above, the following rules for the learning protocol are considered: First, giving the teacher too much power should be avoided, hence the probability distribution over each concept ($F(v) = 1$). This is to avoid allowing the teacher to communicate a program instruction by instruction. Based on the probability distribution,

the protocol must provide some typical examples of vectors for which F is true which is the second important aspect to consider. For if F is true for just one vector that is total, it would require an exponential or more powerful search.

Hence, the learner has access to the following two routines:

1. EXAMPLES(): One each call, provides the learner with a vector v such that F(v) = 1 with the probability D(v)

2. ORACLE(v): Takes a vector v as an input and outputs 1 or 0 based on whether F(v) = 1 or 0.

In the case of learnability, various classes of programs (CNF, DNF and μ Expressions) are examined which have as input a set of Boolean variables $p_1, ..., p_t$ and it is shown that any program can be deduced with only small probability of error in polynomial time. The programs are assumed to take values 1, 0, and undetermined.

As described in [4], a class X of program is *learnable* according to its learning protocol if an algorithm can execute the protocol with the following two properties:

1. Algorithm runs in polynomial time and has an adjustable parameter h which is a singular representation of various parameters that make up the size of program to be learned and some number of variables represented in each vector, t.

2. For all programs in X and distributions D over vectors v for which the output is 1 in regards to X, the algorithm will deduce with probability at least $(1 - h^{-1})$, the observed probability based on program size, a program $g \in X$ which never outputs 1 when it should not but outputs one almost always when it should.

For all vectors v, $g(v) = 1$ implies (i) $f(v) = 1$, and (ii) sum of $D(v)$ over all v such that $f(v) = 1$. On the other hand $g(v) \neq 1$ is at most h^{-1}

In the above protocol, false positives from g are not allowed because the goal is to simply show the possibility of one-sided error learning based on certain program classes. In a more general case, where two-sided error learning is allowed, another probability distribution on the set of vectors for which $f(v) \neq 1$ would be needed. Another aspect of the definition is the use of parameter h in two independent probabilistic bounds for simplification purposes.

A combinatorial bound presented for the probabilistic analysis in different learning expressions is summarized in a single lemma. This is necessary to limit the large solution space in each instance and is achieved through employing heuristics.

As stated in section 4 of [4], "We define the function $L(h, S)$ for any real number h greater than one and any positive integer S as follows: Let $L(h, S)$ be the smallest integer such that in $L(h, S)$ independent Bernoulli trials each with probability at least h^{-1} of success, the probability of having fewer than S successes is less than h^{-1}."

In simpler terms, the function L(h,S) represents the minimum number of trials in which probability of success of each subsequent trial is the unobserved probability at the very least. h represents the size of learning space. This over the total number of trials performed means that the probability of NOT having S successes, the number of expected categories, is less than the total unobserved learning space left.

As a short example, consider an urn containing many different types of marbles. We wish to "learn" what these different types are by taking a sample X. Suppose we also know that there are

only S types of marbles and we wish to choose X such that all but 1 percent of the marbles are represented. L(h,S) implies that if $|X| = L(100, S)$ then with a probability higher than 99 percent we have succeeded in learning about the ratios of S categories in the urn.

3 Learning Expressions

3.1 Bounded CNF-Expressions

A conjunctive normal form (CNF) expression is any product of clauses where each clause is a sum of literals. A literal can either be a variable x or the negation $\neg x$ of the variable. A $k - CNF$ expression for a positive integer k is a $k - CNF expression$ where each clause is the sum of at most k literals. In this section, we regard a clause c_i to be a special kind of expression. Additionally, statements of form $f \Rightarrow g$ or $v \Rightarrow c_i$ can be understood by interpreting the two sides as concepts of functions in an if-then relationship

It is shown that for a fixed k, the class of k-CNF expressions is easily learnable in the sense that no calls of ORACLE are required and EXAMPLES suffice.

Theorem 1 *The class of k-CNF expressions is learnable via an algorithm using $L = L(h, (2t)^{k+1})$ calls of EXAMPLES and no calls of ORACLE, where t is the number of variables in the largest clause.*

The algorithm is initialized with g containing the product of all possible clauses of up to k literals. In the above theorem, $S = (2t)^{k+1}$ because, clearly the number of ways of choosing a clause of k literals is at most $(2t)^k$. Similar to the marble example, this gives us a set of categories from which learning takes place and the study space here is chosen to be greater than what the learner could be presented with based on k number of literals. This sets an upper bound for the number of clauses in g.

Next, the algorithm calls the routine EXAMPLES, L times to gain a representation of concept represented by k-CNF vector v that is total. For each vector obtained, the clauses in g that are not determined to be true in v are deleted. The following is repeated L times.

begin v := *EXAMPLES*
for each c_i *in* g *delete* c_i *if NOT in* v
end

Essentially, learning based on k-CNF expressions can be understood in two steps:

1. Take the total possible space of what could possibly be represented by k literals. This is the total space g that the algorithm begins with.

2. Prune the total space g of the clauses that are not found in examples given by vectors v.

3.2 DNF-Expressions

A disjunctive normal form (DNF) expression is any sum of monomials where each monomial is a product of literals. An expression is monotone if no variable is negated in it. For monotone DNF expressions it is shown that there exists a simple deduction procedure that uses both EXAMPLES

and ORACLE. For unrestricted DNF, a similar result can be proved with respect to a different size measure. In this case, a program can only be deduced for the function, not for the concept. This difference arises from the ability of monotone expression being comparable to the concept by making substitution and asking if the results are identical.

The degree of a DNF-expression is the largest number of monomials that a prime DNF expression equivalent to it can have. For the sake of simplicity, only monotone DNF expressions are discussed here.

Theorem 2 *The class of monotone DNF expressions is learnable via an algorithm B that uses L = L(h,d) calls of EXAMPLES and dt calls of ORACLE, where d is the degree of the DNF expression f to be learned and t the number of variables [4].*

The algorithm is initialized with g = 0. EXAMPLES is then called L times to produce positive examples of each concept. Each time a new vector v is produced such that $v = 1$ does NOT mean $g = 1$, a new monomial m is added to g. The monomial m is the product of those literals determined in v that are essential to making $v = 1$ does not mean $g = 1$ or the concept being positively re-enforced.

This algorithm essentially works through negative learning. The memory or learning space g is 0 at the beginning. Each time vector v produces a negative example that does not exemplify the concept, the specific variables that lead to this negative exemplification are added to g. This means that after the algorithm has gone through the learning space, everything contained in the degree d DNF-expression space that is not in g must be true.

3.3 μ-Expressions

In the previous two subsections, we have seen examples of positive reinforced learning and negative learning. Here we consider a class of functions that can be learned by only using oracles.

Recognizing and parsing expressions with little structural restrictions in comparison to DNF or CNF expressions is seemingly more difficult. This section briefly explores learning through such mechanisms where the oracles need be more sophisticated than those used previously.

A $\mu - expression$ is a monotone expression since in the case of negated values, we can simply re-label them using new variables. Additionally, two $\mu - expressions$ can be regarded identical if their definitions can be made the same by reordering sums and products and relabeling the variables as necessary. In learning of these expressions, no claims regarding the reasonableness of these more powerful oracles are made.

The oracle described in Section 2 is renamed N-ORACLE and N-ORACLE(v)=1 iff for all total vectors w where $w \Longrightarrow v$, it is also the case that $F(w) = 1$. This means that for each vector v, N-ORACLE(v) = 1 only if v agrees with all of the total vectors for which $F(w) = 1$. F is regarded as a Boolean function here, unlike previous sections where it was a concept. Two other oracles are defined here, one is of *relevant possibility RP* and the other of *relevant accompaniment RA*.

The two new ORACLES introduces are defined as follows: $RP(m) = 1$ iff from some monomial m' mm' is a prime implicant of f. Prime implicant is an expression without redundancy in that each variable contained within is unique. The above property states that for each monomial m, if the combination of m and m' is a prime implicant of f, then $RP(m) = 1$. In essence this means, that $RP(m) = 1$ only when completely new information is being brought to the learner, either in comparison to what is already known or two sources of new information.

For sets V, W of variables, $RA(V, W) = 1$ iff every prime implicant of f that contains a variable from V also contains a variable from W. This is self-explanatory, as the ORACLE serves to combine previously known information between two vectors and compare it to already known information.

Theorem 3 *$\mu -$ expressions are learnable through a deduction procedure that uses $O(t^3)$ calls of N, RP and RA ORACLES altogether, where t number of variables are considered.*

To describe the proof briefly, the correct expression is deduced each time through alternately executing a plus-phase and a times-phase using the RA and RP ORACLES respectively. In this way, the newly acquired knowledge by the learner is normalized for the P-ORACLE that compares the newly obtained example in the form of vector v against what is previously known represented by the set of all total vectors w.

4 Cortical Computation Representation and Algorithms

4.1 Cortical Representation

The brain seems to be constrained in the number of ways that a neuron can purposefully effect another neuron and eventually perform some kind of computation. Consider a system of n neurons that receives inputs from a much smaller number of neurons d. There is about a 6 orders of magnitude difference between the two numbers in humans and about 5 orders in mice [3]. In addition, the mean value of excitatory postsynaptic potential (EPSP) of a synapse in a visual cortex has been estimated previously to be .77mV. Compared to an estimate of a typical neuron's threshold voltage of 20mV, this implies the need for about k=26 presynaptic neurons to cause an excitatory EPSP. Another major constraint in cortical computation is that the brain is able to perform computation in the rage of 100-200 milliseconds which allows for about 10-20 steps. In light of this experimental evidence, any basic algorithms tempting to mimic cortical computation need to work in very few steps.

Model of computation discussed here makes use of the three defined parameters (n, d, k), the fact that any basic algorithm would have to run in few steps and that each basic step of the algorithm needs to be simple enough that an actual neuron be able to perform it.

Two representations of information is defined as *shared* or *disjoint*. The representation is *disjoint* if each neuron representing the item is reserved for that sole purpose, or *shared* if different combinations of some neurons represent multiple items.

Some of the shared properties between the two representations are:

1. Representation for each concept is achieved through firing of r neurons
2. Each new concept is at first represented in terms of what is already known.
3. Representation may be statistically graded. To recognize a concept, only a certain fraction of total neurons need to fire.

The issue of neural representations is a widely discussed one. While there is no general rule of thumb that is applicable to cortical computation as a whole, certain areas of the brain seem reserved for certain types of computations. Examples of these are place cells in the hippocampus that map a virtual representation of one's physical surroundings in a distinguishable correlation (OKeefe, et

al., 1998). Determining the exact context in which a neuron fires is much more complicated as there seem to be no general patterns.

The representations discussed in the rest of the paper simplify the representation into a single variable r which represents the density of the representation. The larger the value of r, the more neurons that are required for representation of the corresponding concept. Hence, large values of r mean a more distributed representation.

In terms of the four parameters (n, d, k, r), the following inequality is presented

$$r(d/n) \lesssim k \tag{1}$$

This represents the relation between r active neurons connected by d synapses to n other neurons. This means that each neuron in the set of n neurons receives input, at average, of rd/n inputs. Furthermore, it can be said that k cannot be much less than this quantity, because in that case the number of inputs into n neurons would simply not be sufficient to induce an action potential in the receiving neurons. This would lead to no meaningful computation taking place. The basic idea behind the relationship is that for a total output of r*d innervating n neurons, each neuron on average must be innervated enough (or in the ball part) to induce action potentials.

4.2 Computational Algorithm

The inequality presented in the previous subsection works based on a set of constrains between input vs. output of certain subsets of neurons. Here, an attempt to explain concepts that are represented by networks of neurons in which connections may be few in number, fairly weak and prone to remodeling. A viable candidate are the neurons that represent disjoint information and go under remodeling in the case of hierarchical memory formation. In the case of hierarchical memory formation, new tasks are learned based on what is already known. The already known information is used as foundational blocks for new concepts.

Given two concepts, A and B, a new concept called C must be learned. This means assigning a representing set of neurons to C and changing the circuits as necessary such that if A and B are triggered, the newly allocated neurons for C will also fire. Inversely, it is also important to keep the neurons for C from firing if A or B fire in combination with any other neurons. The proposed mechanism for memory formation can simply be stated as: The neurons representing C are those that are connected to A and B in a manner that sufficient activity at A and B can cause an action potential at each neuron representing C.

The choice of neurons representing C is the result of a probabilistic and random process. Consider the function $B(m, p, s)$ where in m tosses of a coin that comes up heads, with probability p, there will at least be s heads. As s exceeds mp by larger and larger amount, the area where tails (or not heads) is achieved will get smaller at an exponential rate.

If the neurons representing C have at least k connections to the disjoint sets of r neurons representing A and B, then the probability that any arbitrary neuron has these connections to A or B is represented by $B(r, d/n, k)$. If C is considered to be neutral in regards to A and B as it has r representatives in total, then it can reasonably be expected that in that number r, there be n candidate neurons. Hence, the governing equation, as listed in [5] is:

$$(B(r, d/n, k))^2 = r/n \tag{2}$$

In simpler terms, this equation means that k should be sufficently above the value of rd/n such that the resulting probability is r/n. The value of the left side of the equation can be seen to drop

quickly as k increases above the rd/n value. Hence, it follows that equation 1 needs to hold where the total number of inputs that the total number of inputs required to excite a neuron is of the same order of magnitude or a little smaller than rd/n.

5 Conclusions

In this paper, learning was considered as the process of deducing a program to perform a task. This was based on the information that does not provide an explicit description of the program and any means of doing so were prohibited. The notion of learning was given a precise meaning and criteria in which it took place. Namely, this included a learning protocol and a deduction procedure. The learning was restricted to skills that consisted of recognizing whether a concept was true or false for a given data. The concept was considered learned if a program for recognizing it had been deduced. All in all, it can be said that there are specific classes of concepts that are learnable in polynomial time using the learning protocols described.

The latter half of the paper considered a model of cortical computation based on four parameters which were based on the number of outgoing neurons from a system, the neurons innervating the system, the total neurons required for processing a concept and minimum innervations required for EPSPs. Additionally, two general formulas were presented which highlighted the minimum innervations required by a set of neurons based on a general minimum required to perform any kind of meaningful computation. The second equation presented further solidified the conclusions derived by the first using the case of disjoint hierarchical learning.

6 Acknowledgements

Leslie Valiant for his work on computational neuroscience and learning theory. Much of what is presented here comes from that.

References

[1] Probably approximately correct learning - wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Probably_approximately_correct_learning`. (Accessed on 03/12/2016).

[2] Samuel R. Collins. Cogsci summaries. `http://www.jimdavies.org/summaries/valiant1984.html`. (Accessed on 03/12/2016).

[3] Garey Laurence. Cortex: Statistics and geometry of neuronal connectivity 2nd edn. *Journal of Anatomy 194 Pt 1*, 13, March 1999.

[4] L. Valiant. a theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[5] Leslie G. Valiant. A quantitative theory of neural computation. *Biol. Cybern.*, 95(3):205–211, August 2006.