Artificial Intelligence and Novelty

Casey Summers Robinson
University of Minnesota, Morris
caseyr@gmail.com

ABSTRACT

Artificial Intelligence (AI) is a field concerned with developing programs capable of performing functions that typically require human intelligence. A large portion of research done in this field concerns developing programs capable of dealing with novel or previously unseen scenarios, and considering new factors as they are encountered. Some of the biggest challenges in this area are related to the fact that computer programs are incapable of judging what is a "reasonable" action to take without some kind of instruction from a human. We will explore two recently developed models for AI, each of which is capable of handling novel stimuli. One is a recent attempt at implementing a system capable of solving a wide variety of problems. The other is intended for the field of network security, where it is expected to handle threats without first being taught to recognize them.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*knowledge, concept acquisition, induction*

Keywords

Artificial Intelligence, Genetic Algorithms, Artificial Immune Systems

1. INTRODUCTION

Artificial Intelligence (AI) is a field concerned with developing programs capable of performing tasks that typically require human intelligence. Humans are excellent general-purpose problem solvers – given an opportunity to interact with a system about which little is known, and the ability to see the effects that certain actions have upon it, we are capable of generating reasonable inferences about its composition and function.

In contrast to the problem-solving ability of humans, unexpected or novel data are not usually handled gracefully by computer programs. Most programs are not capable of adaptation, and giving these programs the ability to adapt would not necessarily be helpful – a program with consistent behavior is likely to appear more "trustworthy" than one whose behavior may change over time. But in some contexts, the ability to recognize and gracefully handle novel data is more valuable than predictability.

The field of AI has made progress in several subfields, such as pattern recognition [6] and planning [10] . However, individual AI systems (also referred to as AI) tend to be domain-specific; an AI intended for one task is unlikely to perform satisfactorily at another, even if the AI in question is adaptive [8]. This limitation is in part due to the fact that when writing an AI, a programmer typically has a particular goal in mind – a specific problem which it is intended to solve. When writing an AI, a programmer will give it all the tools it needs to solve the problem in question, but those tools are not necessarily sufficient for other problems. It is necessary then for a general problem solver to have the tools with which many problems can be solved, without knowing in advance what these problems may be.

In section 2, we will outline two general machine learning algorithms. Section 3 will describe the process of learning as it relates to artificial intelligence, and finally in section 4, we will look closely at two recent implementations, one of which is intended as a general problem solving model, and the other specific to the field of network security.

2. MACHINE LEARNING ALGORITHMS

The systems we will examine are based upon existing algorithms in the field of AI. In order to understand how these recently developed systems work, some background on their underlying mechanisms is necessary. The two algorithms we will look at are genetic algorithms and artificial immune systems.

2.1 Genetic Algorithms

Genetic algorithms (GAs) are a set of search methods that mimic the process of biological evolution. They start with a large, initially random set of solutions and evaluate them according to a problem-specific *fitness function*, which gives a numerical representation of the quality of the solution. The best solutions from the set are saved, and a new batch of solutions is created by combining traits of the best solutions from the previous generation, introducing a small number of random mutations. By this incremental process, GAs can often generate solutions to extremely complicated problems in a short amount of time [11, 9, 10].

A subfield of GAs is *genetic programming,* which focuses

UMM CSci Senior Seminar Conference, April 2012 Morris, MN.

on the development of computer programs through an evolutionary process. The process by which a new program is generated from those in a previous generation must be carefully handled in order to ensure that the programs generated are syntactically valid. For instance, if the base units being manipulated were the characters which comprise the source code of the program, mutating a semicolon into any other character is likely to yield an invalid program, as semicolons have a specific syntactic function (indicating the end of an expression) which no other character would fulfill. Typically, the smallest units being manipulated are functions.

Some GP implementations operate in a contrived language which lacks many of the strict syntactical features present in most languages, allowing low-level random changes to yield productive features – a process which would be extremely difficult to accomplish in a language such as Java, where a function declaration has certain complex features which are unlikely to come about iteratively.

An example of such a language is Multi-Expression Programming (MEP), which in its most basic form is a series of mathematical expressions with the capacity to use previous expressions as variables. As an example, Figure 2.2 shows an MEP program which computes the average of two inputs, a and b. Lines 3 and 4 use numbers to refer to the values of expressions on the corresponding lines. For instance, on line 4, the expression (3/1) means "the value of the expression on line 3 divided by the value of the expression on line 1."

It is important to note also that no specific line in the MEP program is designated as the solution. This is intentional, as any line in an MEP program can be taken as the result, and which comprises the output is determined only when the program's fitness is being calculated.

Which line is selected as the output of the expression is defined in terms of the solutions for a set of training examples ($S_{0..x}$), and value of the line for these examples ($V_{0..x}$). The line for which $\sum_{i=0}^{x}(|S_i - V_i|)$ is minimized will be taken as the output of the expression for all future uses.

2.2 Artificial Immune Systems

Artificial Immune Systems (AIS) are pattern-matching algorithms which are meant to mimic our own immune system's mechanism of distinguishing between *self* (good) and *non-self* (bad) elements. Biological immune systems tend to attack any foreign material which enters the bloodstream, without first needing to be programmed to recognize this material as "hostile". The immune system attacks this material because the immune system contains a model of the self – material comprising the body, which is not to be attacked – which this foreign material does not fit. AIS are well-suited to contexts where there are many types of behavior which are considered unacceptable, and describing all of them would be cumbersome or impossible.

The core of an AIS is its detector set, as the detectors will determine what the system considers normal. Ideally, no detectors in the set will match data seen in the course of normal operation, and will match any data which is abnormal. The detectors are generated as follows; first, a batch of random detectors is generated, and they are used to examine a large amount of data which comprises the *self* – data that the detectors should not regard as harmful. At this stage, the system selects for those detectors which do not indicate a match with self-data. As the system is selecting for detectors which do *not* present a specified behavior (matching self), this process is known as *negative selection.*

```
1.    a
2.    b
3.  (1+1)
4.  (3/1)
5.  (1+2)
6.  (5/4)
```

Figure 1: An MEP program which finds the average of two numbers, (a,b). Numerical values used in the expression refer to line numbers.

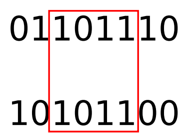

Figure 2: Two differing bitstrings, which have an r-contiguous-bits match for r=4

After detectors undergo negative selection, they can be switched to testing against against a large set of undesirable, or *non-self* data. Here, the system would show preference for those detectors which indicate a match with a high proportion of the data with which they are presented. This stage is known as *positive selection.* From the detectors that are deemed satisfactory, the system constructs a set of detectors that offer the best coverage against the set of known threats, and uses this set to check data being processed by the system for potential threats. In the case that positive selection was not performed, the set of detectors used may be a random set of detectors which passed negative selection.

AISs must be cautious when implementing a set of detectors, as both an overly-strict and overly-relaxed detector set will interfere with the operation of the system. An AIS needs to have gaps in its model of self and non-self in order to allow for the presence of legitimate data which were not used as examples in the negative selection phase. An overfitted detector set will lack these gaps and lead to a large number of false positives, seeing every new piece of data as a threat, while for an underfitted detector set these gaps will be too wide, and the AIS will be unable to generalize to recognize threats upon which it was not trained [3].

Most AIS systems do not depend on matching an entire detector to a piece of data being examined, but instead use a scheme called *r-contiguous bits* matching [5, 3], wherein if over a certain number of bits in a row match between a detector and the data being examined, a match is said to have been found. This is because for detectors of any reasonable size, the probability of a randomly-generated detector completely matching any data considered is very small, and this would severely limit their ability to form a concise detector library.

3. LEARNING

In order for the performance of a learning AI to improve, it must be exposed to a set of problems whose solutions are known, so that the quality of the output generated by the system can be judged by comparing it to ideal solutions. The difference between the system's output and the ideal is referred to as *error,* and the process of adjusting an AI to reduce the error present in its responses to example problems is referred to as *training.*

While training is necessary to adapt an AI to a specific problem, there is always a point at which further training ceases to be beneficial. This occcurs when the system has extracted all the information about the problem that it can from the limited set of training examples, and further training can only serve to improve its performance on these examples. This process is referred to as *overfitting,* and determining when this starts to occur and ceasing training of the system is key to creating systems which can create a general solution to a class of problems from a limited set of examples.

The reason that training on a limited set of problems loses effectiveness over time is that while being trained, the system is modified to minimize the difference between what it produces, and what the training problems specify as correct or acceptable outputs [2]. A system which adequately models the problem will produce answers that are evaluated favorably, but a system which has fit to some feature of the training examples which has no correlation to the original problem could also appear in the training phase to be solving the problem adequately – much as a student who has studied for an exam can perform well, but will be outperformed by a student who has taken the exam several times before and knows that the seventh letter of any given question is its answer, though this does not demonstrate an understanding of the underlying problem.

One method of mediating this problem involves using a separate set of example problems called a *validation set* to judge the quality of the system's solution. The training examples cannot be used for this purpose, because if the system is being trained in a reasonable way, the amount of error in its solutions for these examples is always going to decrease, or stay roughly constant. After each round of training, the amount of error present in the system's responses to the validation set is recorded, and if it has increased after a given training session, this indicates that the system has started to overfit, and training is halted [7]. Halting training before this occurs is also undesirable, as continued training would improve the system's performance on general problems of this type. A system which would have benefited from continued training is called *underfit.*

Once the training data has been used to teach the system, and the validation data used to determine when training ought to be stopped, a third set of example problems must be used in order to determine the quality of the system's solution to the problem. This third set is needed because action has already been taken based upon the system's performance upon the first two, so a set of data upon which the system's performance is unknown is called for.

4. IMPLEMENTATIONS

Now that the basic premises of EC and AIS, as well as the idea of training in a machine learning context, have been explained, the remainder of this paper will be used to examine two recent implementations in the field of AI which use the concepts described to handle novel data.

4.1 General Problem Solving: A-Brain

A-Brain[1] is a general problem solving model proposed by Oltean in his paper [8]. It ties into the ideas of GA and GP discussed earlier, but is intended to autonomously adjust parameters such as population size, frequency of mutations, and allowable maximum solution size, reducing the amount of human input needed to rapidly develop a solution.

4.1.1 Algorithm Description

A-Brain consists of three main components:

- A problem classifier, which determines the nature of the problem to be solved.
- A set of *solvers,* used to generate solutions to identified problems.
- A *trainer,* which is used to generate a new solver when a new type of problem is encountered.

Figure 3 shows a schematic view of these components.

The problem classifier in this scheme has a difficult task. Given only the input for a particular problem, it must determine which of its available solvers (if any) can solve the problem. Oltean's classifier checks the size and type of the input variables, and if an existing solver can take this input, it is assigned to the problem. If this were the only mechanism A-Brain had to classify problems, this would present a significant limit to its generality, as if it learned to solve one problem based upon four integers, it would be unable to learn any other – assigning each new set four integers it was given to the existing "solver which takes four inputs." To avoid this pitfall, problems are also given a short label describing the problem. This way a function which takes four integers and computes their average can exist alongside one which takes four integers and computes their sum, with no risk of mis-classifying one problem as another. In the event that the problem classifier is omitted, but an example problem is included, A-Brain will assign the set of problems it was given to the solver whose output most closely matched that given in the example problem, provided that the error of the output was within a specified tolerance.

If the classifier can find no solver for a problem it is given, it passes control over to the trainer, which prompts the user to input a set of examples – problems paired with their solutions – from which to train a solver. The solver is generated by means of genetic programming. Oltean's implementation uses Multi-Expression Programming (MEP), but he notes any genetic programming approach would suffice, and states that plans exist to create a version of A-Brain which uses neural networks as a solving mechanism.

Since the base fitness function for the solvers being created is similarity of their output to that given in the examples, overfitting is a potential concern here, especially as A-Brain must decide automatically what constitutes an adequate training period. To combat this, the size of a solver is held constant at various stages of the search process, increasing only when no improvements are made over the current best solver for a set number of generations. Oltean's implementation doubles the allowed size of an individual whenever a size increase is called for, and whether this offers an appropriate amount of granularity is debatable. Once a predetermined number of generations have passed with no

[1]Written A$\neq$Brain in the original text

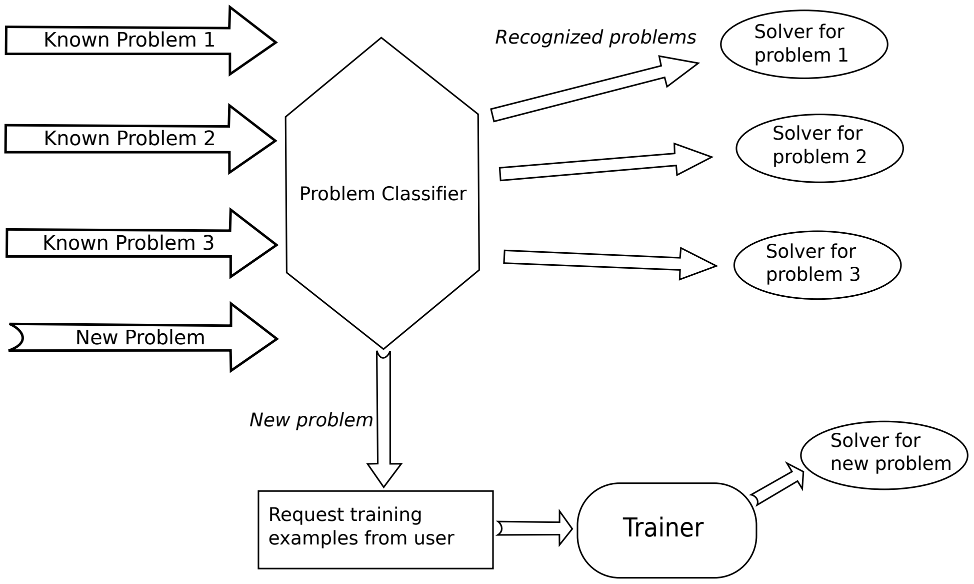

Figure 3: A diagram of A-Brain's operation.

improvement over the previous best found solution, that solution is integrated into the population of solvers, allowing A-Brain to solve problems of this type [8]. It is important to note that while generating a solver, A-Brain is only using evolved MEP individuals, and varying their length. In this sense, A-Brain can be seen as an "autopilot" for standard problem-solving techniques.

4.1.2 Results

Oltean describes testing A-Brain's performance on a small mathematical problem, finding the sum of seven integers, and also several parity problems of varying complexity. Numeric results for all of the problems discussed can be found in table 1. For the sum-seven problem, 1,000 randomly generated datasets were used to train the system over 100 runs.

The remaining problems were of type X-parity – parity problems on bitstrings of length X. A parity problem involves examining a bitstring, and determining whether an even or odd number of bits are set to 1. These parity problems were divided into even-parity and odd-parity classes, so that A-Brain could output a boolean response. As an example, for even-5-parity, the bitstring 01001 would return true, as would 00000, but not 01101 or 00010. For odd-parity, the results for even-parity are reversed – the first two bitstrings would return false, and the last true.

For all experiments, 100 runs were performed for both A-Brain and standard MEP, with the exception of the sum-7 problem which was applied to only A-Brain. For the parity problems, 2^x training cases were given, where x is the length of the bitstrings being considered. That is to say, every possible bitstring of length x, and their solutions, were presented as training data.

For even-3 and 4-parity, the set of boolean functions that A-Brain was allowed to use was restricted to AND, OR, and their negations NAND and NOR, because runs performed with the full set of boolean operators generated perfect solutions too quickly. On these problems, A-Brain achieved

Problem	% Successes		Avg Solution Size	
	Static MEP	A-Brain	Static MEP	A-Brain
Sum-7	omitted	18%	omitted	128
3-Parity	40%	41%	30	78.4
4-Parity	13%	12%	50	480.9
11-Parity	25%	6%	300	256*
12-Parity	80%	3%	500	1024*

Table 1: Table of results from A-Brain and standard MEP. 100 runs were performed for each problem. Entries marked with an * are not averages, but the size of the smallest individual to find a solution.

success rates on par with those realized by multi-expression programming trainers which did not vary the length of solutions over time.

When 11 and 12-parity were considered, the set of expressions available to A-Brain was expanded to encompass all binary operators. Unfortunately, here A-Brain seemed to stumble, achieving far fewer successes than its standard MEP counterpart. Oltean does not offer an explanation for this poor performance in his paper, but it could be due to the fact that structures which contribute to the success of an individual of length 16 will not necessarily have relevance when the size of the individual is increased, but that these structures are carried on to the next generation regardless, and will require extensive crossover and mutation before they are removed completely.

In addition to these simple problems, A-Brain was also used to solve a more complicated problem involving many inputs, and for which a general solution did not already exist. The problem was to determine the amount of electrical power a building would consume based upon the date, time of day, indoor and outdoor temperature, and several other environmental factors. A-Brain was trained against a set of 4,208 data, and created 100 different solvers over 100 runs, having a stated average error of 7.58 and a minimum error of 6.21. Units for this output are never specified, nor is any indication given of how closely this fits the real data. This is regrettable, as an error of 6.21 is insignificant when dealing with quantities in the millions, but quite significant for a range of 0-20. Its effectiveness on data upon which it was not trained also cannot be known, because all available data was used for training, with none being reserved for validation or testing. A competing MEP approach which did not vary the lengths of individuals was also used, and over 100 runs had an average error of 3.81. For that trial, two important notes must be made: the length of individuals in the standard MEP approach was 34 expressions, far fewer than the best solution generated by A-Brain which had a length of 1024 expressions, and only half of the data used to train A-Brain were used in the standard MEP runs, calling into question the validity of the results obtained.

4.2 Security: Artificial Immune Systems

Conventional network security systems are based upon a set of rules, stating which ranges of addresses may be accessed, on what ports and by whom. These rules are typically set by human operators, and must be adjusted manually whenever a change is required. Typically, they fall into two broad categories:

- *Whitelists,* which explicitly grant permissions to certain IP addresses, denying this permission to any address not on the whitelist.
- *Blacklists,* which explicitly deny permissions to certain IP addresses, granting this permission to any address not on the blacklist.

These two categories have flaws. Whitelists can be cumbersome if a large number of addresses are to be allowed to access a service, and blacklists are typically only used to deny permission to specific addresses which have been the source of attacks in the past, which requires identifying the attack and its origin. Blacklists can also be circumvented by an attacker who routes their attack through some computer which is not on the blacklist, concealing the true origin of the attack.

The AIS algorithm offers a way to avoid cumbersome and overly specific rule sets, by generating a concise detector set which is well fitted to the set of behaviors considered normal. In the terms of the rule sets described above, an AIS behaves like a very broad, less draconian blacklist, which flags all behaviors not seen in the normal course of operation as suspicious rather than preventing them outright.

4.2.1 Algorithm Description

To combat overfitting, this AIS model removes detectors from the active set which have failed to form a match for over a specified period of time, replacing them with detectors generated by the negative selection process described in section 2.2.

In addition to conventional r-contiguous-bits detectors, this algorithm also uses some which match on data that has been put through a *permutation mask,* a function which reorders data in a predictable way. An example of such a mask, represented in the form “2-5-4-3-1”, would convert a string “A B C D E” into the string “B E D C A”, by creating a string whose first symbol is the second of the original string, whose second is the fifth of the original, and so on. By reordering the data, certain features which could not have been found by r-contiguous-bits matching on the original due to separation become detectable [4].

4.2.2 Implementation and Results

A paper by Barthrop, et al [4] describes a system called ARTIS, which is an AIS designed to detect network attacks. In the experimental setup, 100 detector sets of 5,000 detectors each were trained, and self-data was provided in the form of eight days of normal network traffic – around 15,000 packets. Positive selection was not performed – the detectors were to regard all non-self data as a potential threat [1]. ARTIS examines a 49-bit representation of network packets, which includes the following information:

- The local IP address associated with the packet
- The remote IP address associated with the packet
- The port along which the packet is travelling (typically indicates the service being accessed by the packet)
- A one-bit flag indicating whether the packet is outbound (intended for the remote IP) or inbound (intended for the local IP)

In the 15,000 packets the system was trained against, only 136 unique strings were seen. This is because ARTIS’s compressed packet representation does not include the large “payload” section of the packet, which contains the information being transferred by the packet. Because of this, communication between one internal address and one external address along a specific port would appear as one unique string, even if thousands of packets each containing different information were sent.

After training, the AIS was exposed to a series of 7,000 packets composed of 426 unique strings, of which 400 represented “attack” data, and 26 were normal network traffic which had been omitted from the original training set. A perfect result would have the system match all 400 attack packets, and fail to match any of the 26 non-threats. The non-permutated sets discriminated relatively poorly, matching $\sim$90% of the attack strings, but also matching a high ($\sim$70%) proportion of the non-threat strings.

The permutated detector sets had very good performance on the test data, with most (97/100) detecting more than 93% of the attack data. Their false-positive rates were more variable, some having as few as 3/26 false positives, and some matching on 24/26. The fact that these detector sets do not have a 100% detection rate on attack packets is potentially problematic, but as Barthrop points out, very few network attacks will generate only one anomalous packet, and so a high but imperfect match rate is acceptable. Likewise, a non-zero false positive rate is also a potential problem, but it is also stated that if the system reacts only when a large number of matches are reported, these false positives will be ignored unless they happen very frequently, which would indicate that the training data for the detector set should be expanded.

5. CONCLUSIONS

A-Brain and AIS represent two very different approaches to handling novel input. AIS attempts to maintain a kind of homeostasis in a computing system, by accepting familiar behavior, even forming the ability to generalize what is "normal", and regarding changes in this behavior as potentially hazardous. The ability to recognize unfamiliar behavior represents a step forward over purely reactionary security systems, which must first be taught about a threat in order to counter it. A-Brain is an interesting approach to evolutionary computation systems, which attempts to reduce the amount of human input needed before a solution can be formed. The fact that it did not outperform standard MEP approaches is in some ways unsurprising, but may be irrelevant because of the use of different training sets for each.

Research in the field of AI is ongoing, and the state of the art changes constantly. These algorithms represent just two recent developments in the field, and their ability to deal with the unexpected represents a noteworthy step forwards.

6. ACKNOWLEDGMENTS

Thanks to Wayne Manselle for his insightful feedback on this paper, and also Nic McPhee and Elena Machkasova for their advice and support through the years, encouraging the right sorts of questions which have led me to where I'm going today, which I hope is somewhere nice.

7. REFERENCES

[1] J. Balthrop, F. Esponda, S. Forrest, and M. Glickman. Coverage and generalization in an artificial immune system. GECCO 2002, pages 3–10, New York, 2002.

[2] Y. Bar-Yam. *Dynamics of Complex Systems.* Westview Press, 2003.

[3] W. Britt, S. Gopalaswamy, J. A. Hamilton, G. V. Dozier, and K. H. Chang. Computer defense using artificial intelligence. In *Proceedings of the 2007 spring simulation multiconference - Volume 3*, SpringSim '07, pages 378–386, San Diego, CA, USA, 2007. Society for Computer Simulation International.

[4] S. A. Hofmeyr and S. A. Forrest. Architecture for an artificial immune system. *Evol. Comput.*, 8(4):443–473, Dec. 2000.

[5] Z. Ji and D. Dasgupta. Revisiting negative selection algorithms. *Evol. Comput.*, 15(2):223–251, June 2007.

[6] N. B. Kalamkar and M. S. Ali. Emotion recognition through facial expression analysis using neuro-fuzzy system. In *Proceedings of the International Conference #38; Workshop on Emerging Trends in Technology*, ICWET '11, pages 719–723, New York, NY, USA, 2011. ACM.

[7] S. Marsland. *Machine Learning: An Algorithmic Introduction.* CRC Press, New Jersey, USA, 2009.

[8] M. Oltean. A-brain: a general system for solving data analysis problems. *Journal of Experimental and Theoretical Artificial Intelligence*, 19(4):333 – 353, 2007.

[9] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming.* Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[10] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, 1995.

[11] H.-T. Wu, W.-T. Hsiao, C.-T. Lin, and T.-M. Cheng. Application of genetic algorithm to the development of artificial intelligence module system. In *Intelligent Control and Information Processing (ICICIP), 2011 2nd International Conference on*, volume 1, pages 290 –294, July 2011.