Peter Henderson

Learning Algorithms to Simulate Human Problem-Solving and Decision-Making

15.11.2011

Abstract:

The following feasibility report compares several machine-learning algorithms to determine which algorithms, or a combination of algorithms, would be the most feasible and best in replicating the human problem-solving and decision-making ability. The study compares two main classes of learning algorithms: supervised and unsupervised. On a more narrow spectrum within these two classes of algorithms, temporal difference learning, Q-learning, the artificial neural network, genetic algorithms and several other smaller decision algorithms are discussed. The study concludes the best class of algorithm to replicate the problem solving capability of a human is the unsupervised learning algorithm, but more specifically, a combination of a neural network and a genetic algorithm.

Introduction:

As technology has progressed through time, it has become prevalently necessary for automated problem solving systems to be created to automatize the maintenance and running of these technologies. To do so successfully, one must replicate the human decision-making ability to solve the problems and obstacles that such autonomous technologies would face. The following report compares several machine-learning algorithms to determine which, or a combination of which, would be the most feasible and best in achieving such a goal. Firstly, it shall compare two classes of learning algorithms, supervised and unsupervised to determine which is the more adept in replicating the human problem-solving skill set. After it has been determined which of these is the more effective class of algorithm, several specific algorithms will be compared from the chosen class to determine which, or a combination of which, is better for dealing with the aforementioned goal. In particular, temporal difference learning, Q-learning, the artificial neural network, and genetic algorithms will be discussed. The study concludes the best class of algorithm to replicate the problem solving capability of a human is the unsupervised learning algorithm, but more specifically, a combination of a neural network and a genetic algorithm.

Background:

In the treatment of such an issue as mimicking the human problem-solving capability, one must first define both machine-learning (including the two main classes of machine-learning algorithms) and break down the human decision-making process. Additionally, the learning algorithms to be compared and discussed shall also be described and defined.

I. Machine-Learning Definition

To define machine learning, one can say "that a machine learns whenever it changes its structure, program, or data (based on its inputs or in response to external information) in such a manner that its expected future performance improves" (Nilsson, 2005). When applied to problem-solving, this can be further elaborated to say that machine learning is when a machine's ability to overcome obstacles and make well-based decisions improves after every subsequent attempt. By analyzing the tries and outcomes of a decision-making process, learning algorithms can be created (and are discussed here) that can successfully replicate the human learning and decision-making process.

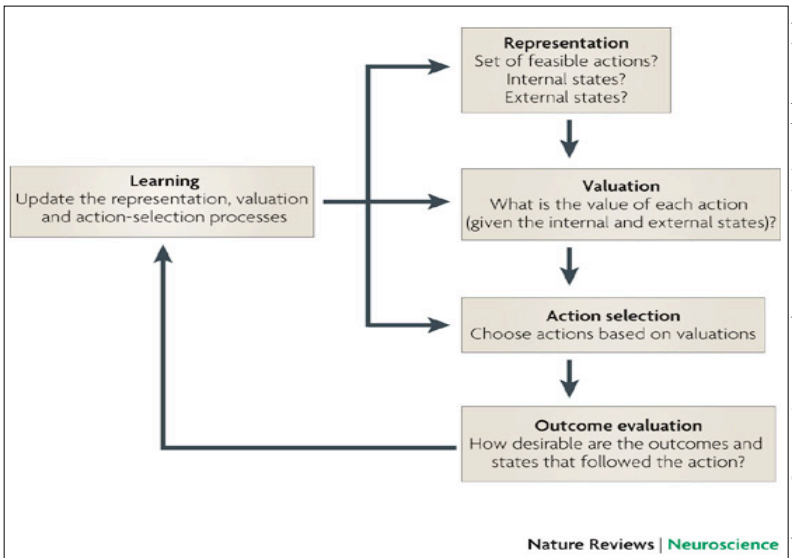

Figure 1: "Value-based decision making can be broken down into five basic processes: first, the construction of a representation of the decision problem, which entails identifying internal and external states as well as potential courses of action; second, the valuation of the different actions under consideration; third, the selection of one of the actions on the basis of their valuations; fourth, after implementing the decision the brain needs to measure the desirability of the outcomes that follow; and finally, the outcome evaluation is used to update the other processes to improve the quality of future decisions." (Rangel

II. The Cognitive Learning Process

This cognitive process in humans, can be broken down into a kind of pseudo-algorithm, that can be closely replicated by computer programming. The study of this break-down and "value-based decision making" inside the human brain is called

neuroeconomics (Rangel, 2008). This can be visualized as seen in Figure 1(Rangel, 2008). The process can be broken down into several main steps and looped to show the learning process. However, one of the most complex parts of writing a learning algorithm is creating a solid "valuation process formula" (Rangel, 2008). Further analysis of the human learning process shows that valuation can be divided into three systems: Pavlovian, habit, and goal-directed. Pavlovian decision systems are the "hard-wired" decision systems in humans: the instinct in making survival decisions. These aren't learned as much as preprogrammed into the human genetic code. Habit systems assign "values to stimulus–response associations (which indicate the action that should be taken in a particular state of the world), on the basis of previous experience, through a process of trial-and-error" (Rangel, 2008). However, due to the fact that these systems rely on "generalization," there is a chance that they might predict the value of actions incorrectly and make incorrect or detrimental decisions. The last and probably the most effective of these systems is the goal-directed valuation system. In contrast with the habit method, it "assigns values to actions by computing action–outcome associations and then evaluating the rewards that are associated with the different outcome... [ideally] the value that is assigned to an action equals the average reward to which it might lead" (Rangel, 2008). To create a truly humanoid problem-solving learning algorithm, one must mimic the habit and goal-directed decision systems successfully so as to construct an artificial intelligence that can learn effectively from past experience to improve solutions and prevent the reuse of faulty solutions.

III. Supervised vs. Unsupervised Learning

In machine-learning there are two classes of algorithms that attempt to recreate this learning system: unsupervised and supervised algorithms. The difference between these two classes of algorithms is that in supervised learning, "the instances are given with known labels (the corresponding correct outputs)... in contrast to unsupervised learning, where instances are unlabeled" (Maglogiannis, 2007). Unsupervised learning algorithms also have a sub-class of "reinforcement learning algorithms"

(Maglogiannis, 2007), where a system learns to predict the risks and rewards of actions by trial-and-error in order to learn to make better decisions from the outcomes. Supervised algorithms have no such function, as the user provides a set of solutions and outcomes and the system strictly follows this labeled set. Though in many cases supervised learning algorithms are useful in machine-learning, particularly in replicating Pavlovian systems where the parameters, or "instincts," are already given, in a truly humanoid problem-solving learning algorithm, it is necessary to be able to encounter new possibilities and instances that don't necessarily fit a given labeled scheme. It is clear that the best-fitting class of algorithm for the aforementioned purpose is the unsupervised algorithm and the reinforcement learning sub-class of algorithms.

IV. Descriptions of Algorithms

Under the unsupervised and reinforcement learning algorithm classes, there is a myriad of algorithms and techniques to try and simulate human learning, however there are several, to be discussed here, that are particularly adept in handling this issue: temporal difference learning, Q-learning, the artificial neural network, and genetic algorithms.

A. Temporal-Difference Algorithms

Temporal difference learning is "a method for approximating long-term future cost as a function of current state" (Tsitsiklis, 1997). At each giving state, a "function approximator" maps a prediction of the cost from going from that state to the future state. "Parameters of the function approximator are updated upon each observation of a state transition and the associated cost. The objective is to improve approximations of long-term future cost as more and more state transitions are observed. The trajectory of states and costs can be generated either by a physical system or a simulated model" (Tsitsiklis, 1997).

B. Q-Learning Algorithms

In Q-learning algorithms (where Q represents the 'state,' or representation of an artificial

intelligences being in relation to what actions it can take), "the agent's experience consists of a sequence of distinct stages or episodes. In the i^{th} episode, the agent: observes its current state X_i, selects [using a mathematical model predicting immediate rewards] and performs an action a_i, observes the subsequent state X_i, receives an immediate reward r_i, and adjusts its Q_{i-1} values" (Nilsson, 2005). Q-learning, unlike temporal-difference learning, does not focus on long term solutions, but rather looks at the immediate reward of an action.

C. Artificial Neural Networks

The artificial neural network (ANN) is another form of structuring an algorithm to optimize decision-making. The ANN is designed to replicate the human brain in its structure so as to maximize processing time as in the human neural network. To do this, the large-scale algorithm behind the ANN is actually composed of many small processing units, each designed to do a specific task (as seen in Figure 2) (PAControl.com, 2011). The combination of these tasks actually "resembles the brain in two respects: [k]nowledge is acquired by the network from its environment through a learning process [and] interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge" (Haykin, 1999). The actual learning process modifies the synaptic weights (or the rankings of importance of the individual processing units and information storage units, the neurons of the ANN), "to attain a desired objective" (Haykin, 1999).

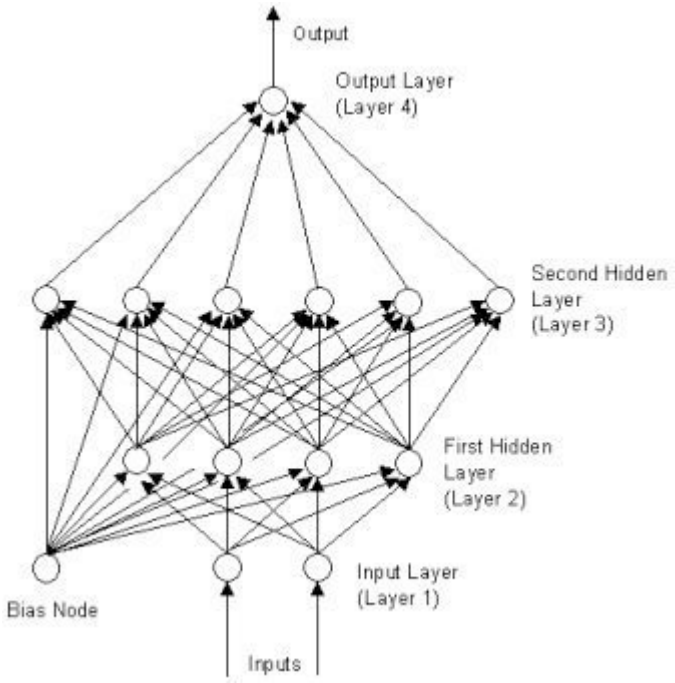

Figure 2: The general outline of what a neural network should look like, being divided into input, data storage, data processing, and weighting "neurons" (Fig. 2: PAControl.com, 2011)

D. Genetic Algorithms

Genetic algorithms are "robust search and optimization techniques" used to find solutions to practical problems. These optimization formulas can be "characterized by the following components: a genetic representation (or an encoding) for the feasible solutions to the optimization problem, a population of encoded solutions, a fitness function that evaluates the optimality of each solution, genetic operators that generate a new population from the existing population, [and] control parameters" (Srinivas, 1994). Just like the evolution of genes in organisms, genetic algorithms create a set of solutions, then through experimentation and evaluation get rid of the "weak" solutions and rank the "stronger" ones higher. Additionally, just like the evolution of a species, genetic algorithms contain crossover and mutation functions that actually create new solutions by mutating and mixing previous solutions. "Crossover causes a structured, yet randomized exchange of genetic material between solutions, with the possibility that 'good' solutions can generate 'better' ones" (Srinivas, 1994). As a result of this, an artificial intelligence system can actually learn to "improvise" new solutions to solve problems.

Discussion and Analysis:

To truly discover which of the described algorithms is the most effective and feasible in replicating human problem-solving and decision-making behavior, the costs and benefits of each must be contemplated and weighed.

I. Temporal-difference Algorithms

Temporal-difference algorithms are both effective and feasible to mimic human problem-solving behaviour. In a research paper produced by IBM, the learning algorithm was analyzed and it was "found that, with zero knowledge built in, the network [was] able to learn from scratch to play the entire game [of backgammon from self-play] at a fairly strong intermediate level of performance... which in fact surpasses comparable networks trained on a massive human expert data set" (Tesauro,

1992). This is a direct example of how the temporal-difference algorithm can be applied successfully to learn to solve problems without human aid. However, though the algorithm looks at long-term rewards to solutions, though perhaps not as precisely and in-depth as other algorithms, it is prone to error because of its method of doing so. This is owed to the fact that by following a pattern of predictions to long-term effects before taking action on any of them, it can actually diverge to infinity from a given solution and loop forever. Though it is one of the most simple and efficient algorithms (in terms of running time and complexity it is $O(n)$[1] as opposed to $O(n^2)$ or $O(n^3)$ in similar functions), "the conditions under which [it] converge[s to an optimal solution] are narrower and less robust than" when using other in other methods (Sutton, 2009a). As such, it would be better to look at those other methods before considering TD algorithms as a solution to achieve the given goal.

II. Q-Learning Algorithms

Though the Q-Learning algorithm is more complex (with a complexity ranging from $O(n)$ to $O(n^3)$ in the worst case scenario (Koenig, 1993)) than the temporal-difference algorithm, it has still been used effectively and perhaps with more success since it is "typically easier to implement" (Kaelbling, 1996). This gives it a better opportunity to run without crashing and to come up with the optimal solutions with a greater probability than Temporal-difference (TD) learning algorithms (Kaelbling, 1996). The Q-learning method is actually relatively similar to the TD method in its construct, though it is more greedy in its usage of memory and is exploration insensitive[2]. However, because of its exploration insensitivity, it often makes for the most effective algorithm, since it will almost always run and converge to a proper solution if implemented correctly – unlike with the temporal-difference algorithm, which still looks somewhat at the long term due to it being exploration

1 $O(n)$ is a representation of the complexity of an algorithm, based on the number of operations it does, the greater the function inside the O(), the more complex it is and the greater running time it has.

2 *Exploration insensitive* means that that the "Q values will converge to the optimal values, independent of how the agent behaves while the data is being collected" (Kaelbing, 1996). In other words, the Q-learning algorithm is insensitive to looking at the long term benefits of certain solutions and will immediately discard them if they do not provide the greater short-term benefit.

sensitive, resulting in a chance of non-convergence and malfunctioning (Kaeling, 1996). Due to the TD algorithm's possibility of failure in certain cases, it can be discarded as a solitary solution to the presented issue, yet the Q-learning algorithm can also be discarded due to its lack of analysis of the long term benefits of various solutions. As such, attention must be turned to the most efficient and most human-like algorithms in their ability to analyze solutions, make decisions, and solve problems.

III. Artificial Neural Networks

The first of these human-based algorithms is the Artificial Neural Network. As previously described, this type of algorithm is actually divided into many small processing units, or neurons. This makes the complexity of such an algorithm varied according to the complexity of each neuron. Due to the system's ability to divide functionality and work among various units, this easily allows for the possibility of an algorithm of complexity O(n) or even less (Gupta, 2011). This functionality makes it an ideal choice for actual implementation due to its ability to quickly process information. Additionally, because each neuron is weighted and can be altered in its priority, this allows for the distribution and adjustment of complexity. If the system were to become overloaded, certain neurons of lower priority can be assigned to be processed later in time, or not at all. Due to this capability, several major applications, mimicking the human problem-solving capacity, have already been implemented into prototype systems. Two such examples are the autonomous robot driving vehicle and the Honda Asimo automated robotic learning system. The autonomous robot driving vehicle, proposed in Pomerleau's (1993) paper, is directly based on a neural network called “ALVINN (Autonomous Land Vehicle In a Neural Network).” The paper “presents the neural network architecture and training techniques that allow ALVINN to drive in a variety of circumstances including single-lane paved and unpaved roads, multilane lined and unlined roads, and obstacle-ridden on- and offroad environments, at speeds of up to 55 miles per hour” (Pomerleau, 1993). The Honda Asimo robot, in a system similar to that of Manoonpong's (2007) paper, also uses a neural network to adaptively solve problems in its

walking path as well as to learn to recognize objects in an environment from a video feed. Clearly, the power of neural networks can be seen in these examples. The problems solved by neural networks are incredibly similar to those needing to be solved by humans on a daily-basis, and the neural network allows these systems to do so in a similarly efficient fashion. The only cost of using an ANN is the fact that a large training set has to be given to the system so it can learn which solutions to avoid and which to use. However, this is similar to the capabilities of humans. A human has many years of training and thousands of training sets given to them as children to develop his or her problem solving and decision-making ability. Additionally, due to the flexibility of this type of system, other algorithms can be implemented to actually increase the efficiency and capabilities of the ANN, to allow them to evolve and adapt on their own at an even quicker pace than humans.

IV. Genetic Algorithms

One such incredibly powerful tool in improving ANNs is the genetic algorithm. A genetic algorithm actually gives the ANN the capability of evolving its decision-making skill-set over time. Since a genetic algorithm has the ability to find the global optimum of a system and process multimodal functions, it is ideal for a humanoid decision system as it can look at the long term solutions as well as the short term as well as processing multiple types of solutions and problems at once. In fact, in today's neural networks, genetic algorithms are already used to adjust the weights and priority of each neuron in an optimal fashion (Srinivas, 1994). Additionally, due to their relatively low complexity (between O(n) and $O(n^2)$, depending on if the data grows exponentially or on a linear function), it is not incredibly costly to put them incorporate them into a neural network, especially when they actually help decrease the complexity of a neural network by optimizing neural weights (Lobo, 2000).

Conclusion:

As seen in the previous analysis, the capabilities of the neural network and genetic algorithms

are ideal for replicating the human decision and problem-solving processes. The power of the neural network, especially in its ability to "divide-and-conquer" problems, allows it to process solutions quickly, rearranging itself to best fit the scenario. In combination with a genetic algorithm, which is relatively simple in terms of complexity, the neural network is able to efficiently weigh solutions based on both long and short term solutions and to find the universal optimum in nearly any situation. Though, with an exponentially large set of data, these systems, could crash, given today's technology and a filter put on the data flow, it is already feasible to implement these algorithms into real-world systems, as seen in the given examples of the autonomous car and the adaptive robot, implementing both visual learning and obstacle avoidance[3].

I. Recommendations For Future Applications and Research

With the exponential growth in the capacity and efficiency of technology according to Moore's law, it is possible, with further modification and research of these types of algorithms, to expand the functionality and capability of already existing autonomous systems, and eventually to create truly autonomous intelligent systems. One specific application of the genetically-modified neural network is to create an intelligent agent capable of taking in and analyzing linguistic patterns to learn how to "understand" and respond to dialogue. This would be a large leap in the functionality of these algorithms, but it is definitely possible with the right processing procedures inside the ANN and a large enough processing unit. A specific implementation of the genetically modified ANN to such a purpose would be to create several dedicated neural systems to take in input in the form of speech or text input, break it down to phrases (or subjects, objects, and verbs), cross-reference the broken down material against proper responses, genetically cross possible solutions, and weigh the best solution based on

3 For videos of prototypes implementing these algorithms see the following:
http://www.youtube.com/watch?v=YPoANTKo5kA&feature=related
http://www.youtube.com/watch?v=P9ByGQGiVMg
http://www.youtube.com/watch?v=-nYhKD8leAg&feature=related

sentence structure and relevance to the input.

Conclusively, though true Artificial Intelligence systems do not yet exist, learning algorithms such as a genetically modified neural network, with further research, make the goal of replicating human decision-making and problem-solving behaviour a real possibility with today's ever-growing technologies.

Sources:

Gupta, P., & Poonacha, P.G. (2011). Depth-optimal O(n)-node Neural Networks for n-bit addition. Bombay, India. Indian Institute of Technology.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*. *The Knowledge Engineering Review* (Vol. 13, pp. 409-412). Prentice Hall. Retrieved from http://www.journals.cambridge.org/abstract_S0269888998214044

Kaelbling, L.P., Littman, L.M., & Moore,A.W. (1996). Reinforcement learning: A survey *J. Artif. Intell. Res.*, vol. 4, pp. 237–285

Koenig, S., & Simmons, R.G. (1993). Complexity analysis of real-time reinforcement learning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence,* pp.99-105 Menlo Park, California. AAAI Press/MIT Press.

Lobo, F. G., Goldberg, D.E., & Pelikan, M. (2000).Time complexity of genetic algorithms on exponentially scaled problems. In GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference, pages 151--158. Morgan Kaufmann.

Maglogiannis, I. G. (2007). Emerging artificial intelligence applications in computer engineering: real word AI systems with applications in eHealth, HCI, information retrieval and pervasive technologies, IOS Press.

Nilsson, N. J. (2005). *Introduction to machine learning*. Stanford: Stanford University.

PAControl.com. (2011). Artificial neural networks. Retrieved from http://www.pacontrol.com/Neural.html

D.A. Pomerleau, Knowledge-based training of artificial neural networks for autonomous robot driving, J.H. Connell, S. Mahadevan, Editors , *Robot Learning*, Kluwer Academic Publishers, Dordrecht (1993), pp. 19–43

Rangel, A., C. Camerer, et al. (2008). "A framework for studying the neurobiology of value-based

decision making." Nat Rev Neurosci 9(7): 545-556.

Srinivas, M.; Patnaik, L.M. (1994). "Adaptive probabilities of crossover and mutation in genetic algorithms," *Systems, Man and Cybernetics, IEEE Transactions on* , vol.24, no.4, pp.656-667
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=286385&isnumber=

Sutton, R. S., H. R. Maei, et al. (2009a). Fast gradient-descent methods for temporal-difference learning with linear function approximation. Proceedings of the 26th Annual International Conference on Machine Learning. Montreal, Quebec, Canada, ACM: 993-1000.

Sutton, R. S. (2009b). "A convergent O (n) algorithm for off-policy temporal-difference learning with linear function approximation." Advances in neural information processing systems **21**: 1609.

Tesauro, G. (1992). "Practical issues in temporal difference learning." Machine Learning 8(3): 257-277.

Tsitsiklis, J. N. and B. Van Roy (1997). "An analysis of temporal-difference learning with function approximation." Automatic Control, IEEE Transactions on 42(5): 674-690.