ANALYSIS OF AGING-WEARING FOR MILITARY VEHICLES BY USING MACHINE LEARNING ALGORITHMS

Middle East Technical University
Computer Enginering Department
Ankara
Nov,2003

Abstract. The subject of this study is to construct a machine learning based system that estimates the aging/wearing properties of land vehicles. A hybrid approach combining both inductive and analytical machine learning methods is to be used since both training data and domain theory is expected to be available. This paper initially describes knowledge-based neural networks and then continues to discuss two extensions of it, the TopGen and REGENT algorithms. Both the basic aspects and the details of these algorithms are reviewed to infer their advantages and disadvantages. Finally, the possible outcomes and corresponding steps to take are mentioned after the application of the methods.

Table of Contents

1. Introduction

This report aims to describe a specific analysis that is a part of a military project whose subject is modelling and simulation of land vehicles. The analysis under consideration aims to estimate aging-wearing properties of land vehicles. This estimation needs automation since military forces have a large database containing past information about land vehicles. Altitude, climatic conditions, road conditions, being shot by enemy forces, vehicle parameters such as acceleration, fuel usage, duration of movement, etc and the eventual aging/wearing results belonging to each land vehicle in a certain period are all listed as seperate records in this information. This huge pile of knowledge will be interpreted by a learning-based system. In the past, statistical methods are more prevalently used to extract facts or guesses from such databases. However, since the database for the current analysis contains rather scarce and erroneus data, statistical methods are more subject to failure than the machine learning algorithms.

This paper discusses three machine-learning algorithms, which are KBANN, Topgen and REGENT. KBANN is a well-known method that is described in the background section, whereas the latter ones are extensions of KBANN that introduce further improvements over KBANN. The discussion section tries to infer which model appears to be more beneficial. The conclusion part addresses how these methods can be applied to the current project.

2. Background

2.1 KBANN: An Inductive-Analytical Method in Machine Learning

Inductive methods seek general hypotheses that fit observed training data. They can fail with insufficient data and may be misled by incorrect bias. Analytical methods seek general hypotheses that fit observed training data and prior knowledge. They can generalize more accurately from less data and can be misled with insufficient prior data. Combining approaches offers possibility of powerful learning methods such as KBANN, knowledge-based neural networks.

KBANN are networks whose topology is determined by mapping the dependencies of a domain-specific rule base into a neural network. After the neural network is initialized to predict domain theory, it is then refined to fit the training data.

To achieve this, we firstly create a sigmoid unit for each Horn clause in the domain theory. If sigmoid output is greater than 0.5, it is interpreted as true. If sigmoid output is less than 0.5, it is interpreted as false. Input is created for each antecedent.

Weights are set to compute logical AND of the inputs. For each input corresponding to a non-negated antecedent, weights are set to positive constant W. For each input corresponding to a negated antecedent, weights are set to negative W.

Each unit is constructed so that output will be greater than 0.5 just for cases for its Horn clauses. Threshold weight of the unit w0 is set to – (n-0.5) W. With 0 and 1 inputs, correct output is guaranteed. Additional input units are added to each threshold unit and weights are set approximately to 0. After the initial neural network is constructed, we refine the network using inductive learning. The tuning process learns new dependencies.

KBANN generally generalizes more accurately than pure back-propagation, especially with scarce data. Methods have been developed for mapping the refined network back to Horn clauses. KBANN has been shown to be more effective at classifying previously-unseen examples than a wide variety of machine learning algorithms [Towell et al., 1990; Towell, 1992].

A large part of the reason for KBANN's superiority over other symbolic systems has been attributed to both its underlying learning algorithm (i.e., backpropagation) and its effective use of domain-specific knowledge [Towell, 1992].

3. Related Work

3.1 Heuristically Expanding KBANN: The TopGen Algorithm

Existing network training methods for KBANN lack the ability to add new rules to the rule bases. Thus, on domain theories that are lacking rules, generalization is poor, and training can corrupt the original rules, even those that were initially correct. A presented extension is TopGen [Opitz and Shavlik, 1993], an extension to the KBANN algorithm that heuristically searches for possible expansions of a knowledge-based neural network, guided by the domain theory, the network, and the training data. It does this by dynamically adding hidden nodes to the neural representation of the domain theory, in a manner analogous to adding rules and conjuncts to the symbolic rule base.

TopGen uses a symbolic interpretation of the trained network to help decide where the primary errors are in the network. Units are added in a matter analogous to adding rules and conjuncts to the symbolic rulebase. Adding hidden nodes in this fashion synergistically combines the strengths of refining the rules symbolically with the strengths of refining them with backpropagation.

TopGen heuristically searches through the space of possible ways of adding nodes to the network, trying to find the network that best refines the initial domain theory. Briefly, TopGen looks for nodes in the network with high error rates, and then adds new nodes to these parts of the network. TopGen uses two tuning sets, one to evaluate the different network topologies, and one to help decide where new nodes should be added (the latter tuning set is also used to decide when to stop training individual networks). TopGen uses KBANN's rule-to-network translation algorithm to define an initial guess for the network's topology. TopGen trains this network using backpropagation [Rumelhart et al., 1986] and places it on a search queue. In each cycle, TopGen takes the best network from the search queue (as measured by tuning-set-2), decides possible ways to add new nodes, trains these new networks, and places them on the search queue. This process repeats until reaching either (a) a tuning-set-2 accuracy of 100% or (b) a previously-set time limit.

TopGen:
GOAL: Search for the best network describing the domain theory and training examples.
1. Set aside a testing set. Break the remaining examples into a training set and two tuning sets (tuning-set-1 and tuning-set-2).
2. Place the trained network, produced by KBANN, on the search queue.
3. Until stopping criteria met:
(a) Remove the best network, according to tuning-set-2, from the search queue.
(b) Use ScoreEachNode to determine the N best ways to expand the topology.
(c) Create N new networks, train and put on the search queue.
(d) Prune search queue to length M.
4. Output the best network seen so far according to tuning-set-2.

ScoreEachNode:
GOAL: Use the errors in tuning-set-1to suggest good ways to add new nodes.
1. Set each node's correctable-false-negative and correctable-false-positive counters to 0. Assume each node is a threshold unit.
2. For each misclassified example in tuning-set-1, consider each node and determine if modifying its output will correctly classify the example, incrementing the counters when appropriate.
3. Use the counters to order possible node corrections. High correctable-false-negative counts suggest adding a disjunct while high correctable-false-positive counts suggest adding a conjunct.

Table 1. The TopGen Algorithm

3.1.1 Where Nodes Are Added

TopGen must first find nodes in the network with high error rates. It does this by scoring each node using examples from tuning-set-1. By using examples from this tuning set, TopGen adds nodes on the basis of where the network fails to generalize, not where it fails to memorize the training set. TopGen makes the empirically-verified assumption that almost all of the nodes in a trained knowledge-based network are either fully active or inactive. By making this assumption, each non-input node in a TopGen-net can be treated as a step function (or a Boolean rule) so that errors have an all-or-nothing aspect to them. This concentrates topology refinement on misclassified examples, not on erroneous portions of each example.

TopGen keeps two counters for each node, one for false negatives and one for false positives defined with respect to each individual node's output. TopGen increments counters by recording how often changing the ``Boolean" value of a node's output leads to a misclassified example being properly classified. That is, if a node is active for an erroneous example and changing its output to be inactive results in correct classification, then TopGen increments the node's false-positives counter. TopGen increments a node's false-negatives counter in a similar fashion. By checking for single points of failure, TopGen looks for rules that are near misses. TopGen adds nodes where counter values are highest, while breaking ties by preferring nodes farthest from the output node.

3.1.2 How Nodes Are Added

Once we know where to add new nodes, we need to know how to add these nodes. TopGen makes the assumption that when training one of its networks, the meaning of a node does not shift significantly. Making this assumption allows to alter the network in a fashion similar to refining symbolic rules. Towell [1992] showed that making a similar assumption about KBANN-nets was valid. In a symbolic rulebase that uses negation-by-failure, we can decrease false negatives by either dropping antecedents from existing rules or adding new rules to the rulebase. Since KBANN is effective at removing antecedents from existing rules, TopGen adds nodes, intended to decrease false negatives, in a fashion that is analogous to adding a new rule to the rulebase. If the existing node is an OR node, TopGen adds a new node, fully-connected to the inputs, as its child. If the existing node is an AND node, TopGen creates a new OR node that is the parent of the original AND node and another new node that TopGen fully-connects to the inputs; TopGen moves the outgoing links of the original node to become the outgoing links of the new OR nodes. To decrease false positives in a symbolic rulebase, we can either add antecedents to existing rules or remove rules from the rulebase. While KBANN can effectively remove rules [Towell, 1992], it is less effective at adding antecedents to rules and is unable to invent (constructively induce) new terms as antecedents.

By allowing these additions, TopGen is able to add rules whose consequents were previously undefined to the rulebase. TopGen handles nodes that are neither AND nor OR nodes by deciding if such a node is closer to an AND node or an OR node (by looking at the node's bias and incoming weights). TopGen classifies previously-added nodes in such a manner, when deciding how to add more nodes to them at a later time.

3.1.3 Additional Algorithmic Details

After new nodes are added, TopGen must train the network. While we want the new weights to account for most of the error, we also want the old weights to change if necessary. That is, we want the older weights to retain what they have previously learned, while at the same time move in accordance with the change in error caused by adding the new node. In order to address this issue, TopGen multiplies the learning rates of existing weights by a constant amount (≤ 1) every time new nodes are added, producing an exponential decay of learning rates.

To help address the trade-off between changing the domain theory and disregarding the misclassified training examples as noise, TopGen uses a variant of weight decay [Hinton, 1986]. Weights that are part of the original domain theory decay toward their initial value, while other weights decay toward zero. Thus, we add to the usual cost function, a term that measures the distance of each weight from its initial value:

$$Cost = \sum_{k \in T} (target_k - output_k)^2 + \lambda \sum_{i \in W} \frac{(\omega_i - \omega_{init_i})^2}{1 + (\omega_i - \omega_{init_i})^2}$$

The first term sums over all training examples T, while the second term sums over all weights W. The tradeoff between performance and distance from initial values is weighted by λ.

3.2 Genetically Searching the Space of Network Topologies: The Regent Algorithm

Another algorithm proposed by Opitz and Shavlik, REGENT [Opitz and Shavlik, 1997], tries to broaden the types of networks that TopGen considers with the use of GAs. We can view REGENT as having two phases: (a) genetically searching through topology space, and (b) training each network using backpropagation's gradient descent method. REGENT uses the domain theory to aid in both phases. It uses the theory to help guide its search through topology space and to give a good starting point in weight space.

Table 2 summarizes the REGENT algorithm. REGENT first sets aside a validation set (from part of the training instances) for use in scoring the different networks. It then perturbs the KBANN-produced network to create an initial set of candidate networks. Next, REGENT trains these networks using backpropagation and places them into the population. In each cycle, REGENT creates new networks by crossing over and mutating networks from the current population that are randomly picked proportional to their fitness (i.e., validation-set correctness). It then trains these new networks and places them into the population. As it searches, REGENT keeps the network that has the lowest validation-set error as the best concept seen so far, breaking ties by choosing the smaller network in an application of Occam's Razor. A parallel version of REGENT trains many candidate networks at the same time using the Condor system [Litzkow, Livny, & Mutka 1988], which runs jobs on idle workstations.

GOAL: Search for the best network topology describing the domain theory and data.

1. Set aside a validation set from the training instances.
2. Perturb the KBANN-produced network in multiple ways to create initial networks, then train these network using backpropagation and place them into the population.
3. Loop forever:
    a. Create new networks using the crossover and mutation operators.
    b. Train these networks with backpropagation, score with the validation set, and place into the population.
    c. If a new network is the network with the lowest validation-set error seen so far(breaking tics by preferring the smallest network), report it as the curent best concept.

Table 2: The REGENT algorithm.

A diverse initial population will broaden the types of networks REGENT considers during its search; however, since the domain theory may provide useful information that may not be present in the training set, it is still desirable to use this theory when generating the initial population. REGENT creates diversity around the domain theory by randomly perturbing the KBANN network at various nodes. REGENT perturbs a node by either deleting it, or by adding new nodes to it in a manner analogous to one of TopGen's four methods for adding nodes. (Should there happen to be multiple theories about a domain, all of them can be used to seed the population.)

3.2.1 REGENT's Crossover Operator

REGENT crosses over two networks by first dividing the nodes in each parent network into two sets, `A` and `B`, then combining the nodes in each set to form two new networks (i.e., the nodes in the two `A` sets form one network, while the nodes in the two `B` sets form another). Table 3 summarizes REGENT's method for crossover and Figure 1 illustrates it with an example.

REGENT divides nodes, one level at a time, starting at the first level, which is nearest the output nodes. When considering a level, if either set `A` or set `B` is empty, it cycles through each node in that level and randomly assigns it to either set. If neither set is empty, nodes are probabilistically placed into a set.

Crossover Two Networks:
GOAL: Crossover two networks to generate two new network topologies.

1. Divide each network's hidden nodes into sets A and B using **DivideNode**.
2. Set A forms one network, while set B forms another. Each new network is created as follows:
    - **a.** A network inherits weight w_{ji} from its parent if nodes i and j either are also inherited or are input or output nodes.
    - **b.** Link unconnected nodes between levels with near-zero weights.
    - **c.** Adjust node biases to keep original AND or OR function of each node

DivideNodes:
GOAL: Divide the hidden nodes into sets A and B, while probabilistically maintaining each network's rule structure. While some hidden node is not assigned to set A or B:

```
(i)     Collect those unassigned hidden nodes whose output is linked only to either
        previously-assigned nodes or output nodes.
(ii)    If set A or set B is empty:
                For each node collected in part (i), randomly assign it to set A or set B.
        Else
                Probabilistically add the nodes collected in part (i) to set A or set B.
```

Table 3: REGENT's method for crossing over networks.

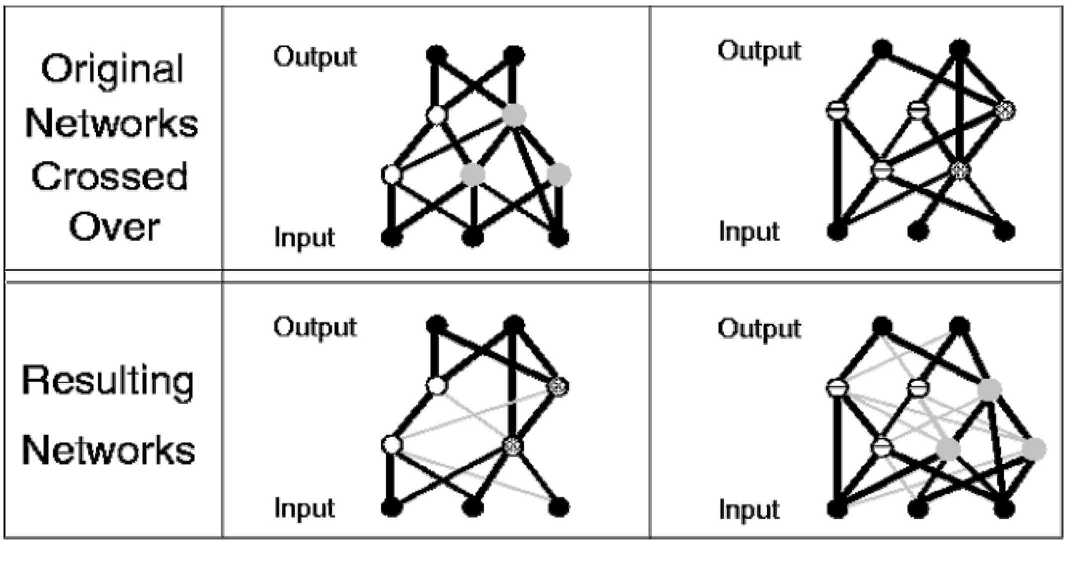

Figure 1: REGENT'S method for crossing over two networks.

An equation calculates the probability of a given node being assigned to set `A`. The probability of belonging to set `B` is one minus this probability. With these probabilities, REGENT tends to assign to the same set those nodes that are heavily linked together. This helps to minimize the destruction of the rule structure of the crossed-over networks, since nodes belonging to the same syntactic rule are connected by heavily linked weights. Thus, REGENT's crossover operator produces new networks by crossing-over rules, rather than simply crossing-over nodes.

REGENT must next decide how to connect the nodes of the newly created networks. First, a new network inherits all weight values from its parents on links that (a) connect two nodes that are both inherited by the new network, (b) connect an inherited hidden node and an input or output node, or (c) directly connect an input node to an output node. It then adds randomly set, low-weighted links between unconnected nodes on consecutive levels.

Finally, it adjusts the bias of all AND or OR nodes to help maintain their original function. For instance, if REGENT removes a positively weighted incoming link for an AND node, it decrements the node's bias by subtracting the product of the link's magnitude and the average activation (over the set of training examples) entering that link. This is done since the bias for an AND node needs to be slightly less than the sum of the positive weights on the incoming links. REGENT increments the bias for an OR node by an analogous amount when it removes negatively weighted incoming links (since the bias for an OR node should be slightly greater than the sum of the negative weights on the incoming links so that the node is inactive only when all incoming negatively weighted linked nodes are active and all positively weighted linked nodes are inactive).

3.2.2 REGENT's Mutation Operator

REGENT mutates networks by applying a variant of TopGen. REGENT uses TopGen's method for incrementing the false-negatives and false-positives counters for each node. REGENT then adds nodes, based on the values of these counters, the same way TopGen does. Since neural learning is effective at removing unwanted antecedents and rules from KBNNs, REGENT only considers adding nodes, and not deleting them, during mutation. Thus, this mutation operator adds diversity to a population, while still maintaining a directed, heuristic-search technique for choosing where to add nodes; this directedness is necessary because we currently are unable to evaluate more than a few thousand possible networks per day.

3.2.3 Additional Details

REGENT adds newly trained networks to the population only if their validation-set correctness is better than or equal to an existing member of the population. When REGENT replaces a member, it replaces the member having the lowest correctness (ties are broken by choosing the oldest member). Other techniques such as replacing the member nearest the new candidate network, can promote diverse populations; however, one wouldn't not want to promote diversity at the expense of decreased generalization.

REGENT can be considered a Lamarckian (Lamarckian evolution is a theory based on the inheritance of characteristics acquired during a lifetime.), genetic-hillclimbing algorithm, since it performs local optimizations on individuals, then passes the successful optimizations on to offspring. The ability of individuals to learn can smooth the fitness landscape and facilitate subsequent learning. Thus, Lamarckian learning can lead to a large increase in learning speed and solution quality.

4. Discussion

An ideal inductive-learning algorithm should be able to exploit the available resources of extensive computing power and domain-specific knowledge to improve its ability to generalize. KBANN has been shown to be effective at translating a domain theory into a neural network; however, KBANN suffers in that it does not alter its topology. The TopGen algorithm outperforms KBANN since it has the advantage to modify the network topology. However, it also risks the possibility that the weight updates may wipe out the initial domain theory, especially in cases where the examples are drawn from a subset of the entire domain theory.

TopGen showed statistically-significant improvements over KBANN in several real-world domains, and comparative experiments with a simpler approach to adding nodes verified that new nodes must be added in an intelligent manner [Opitz and Shavlik, 1993]. However, when we increase the number of networks TopGen considers during its search, it is observed that the increase in generalization is primarily limited to the first few networks considered [Opitz and Shavlik, 1997]. Therefore, TopGen is not so much an ``anytime'' algorithm, but rather is a first step towards one. This is mostly due to the fact that TopGen only considers larger networks that contain the original KBANN network as subgraphs; however, as one increases the number of networks considered, one should also increase the variety of networks considered during the search. Therefore, TopGen lacks broadening the range of networks considered during the search through topology space and unable to improve its performance after searching beyond a few topologies.

A new algorithm, REGENT, uses a specialized genetic algorithm to broaden the types of topologies considered during TopGen's search. Experiments indicate that REGENT is able to significantly increase generalization over TopGen; hence, this new algorithm is successful in overcoming TopGen's limitation of only searching a small portion of the space of possible network topologies. In doing so, REGENT is able to generate a good solution quickly, by using KBANN, then is able to continually improve this solution as it searches concept space. Therefore, REGENT takes a step toward a true anytime theory refinement system that is able to make effective use of problem-specific knowledge and available computing cycles.

5. Conclusion and Future Work

The TopGen and REGENT algorithms generally produce better results than KBANN with an initial propositional domain theory and training data since they also change the topology of the initial neural network.. In addition, REGENT is better than TopGen since it searches a wider range of topologies due to its genetic algorithm.

After a simple domain theory about aging/wearing properties of land vehicles is constructed by consulting the experts, all the three methods will be applied as a benchmark study. If the initial domain theory turns out to be more complex with the usage of variables, then an inductive-analytical approach that allows first-order logic in its domain theory may be applied. Or if it is realized that domain theory is too inefficient to make use of, purely inductive methods will also be considered.

6. References

1. [Towell et al., 1990] G. Towell, J. Shavlik, and M. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In Proc. of the 8th Nat. Conf. on Artificial Intelligence, pages 861--866, Boston, MA, 1990.

2. [Towell, 1992] G. Towell. Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction. PhD thesis, Univ. of Wisconsin, Madison, WI, 1992.

3. [Opitz and Shavlik, 1993] David W. Opitz and Jude W. Shavlik.
Heuristically Expanding Knowledge-Based Neural Networks. In Proc. of the 13th International Joint Conf. on Artificial Intelligence (IJCAI-93).

4. [Rumelhart et al., 1986] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In D. Rumelhart and J. McClelland, editors, Parallel Distributed Processing, Volume 1. MIT Press, Cambridge, MA, 1986.

5. [Hinton, 1986] G. Hinton. Learning distributed representations of concepts. In Proc. of the 8th Annual Conf. of the Cognitive Science Soc., pages 1--12, 1986.

6. [Opitz and Shavlik, 1997] David W. Opitz and Jude W. Shavlik. Connectionist Theory Refinement: Genetically Searching the Space of Network Topologies. Journal of Artificial Intelligence Research.

7. [Litzkow, Livny and Mutka, 1988] Litzkow, M., Livny, M., Mutka, M. 1988. Condor -- a hunter of idle workstations. In Proceedings of the Eighth International Conference on Distributed Computing Systems, 104-111, San Jose, CA. Computer Society Press.